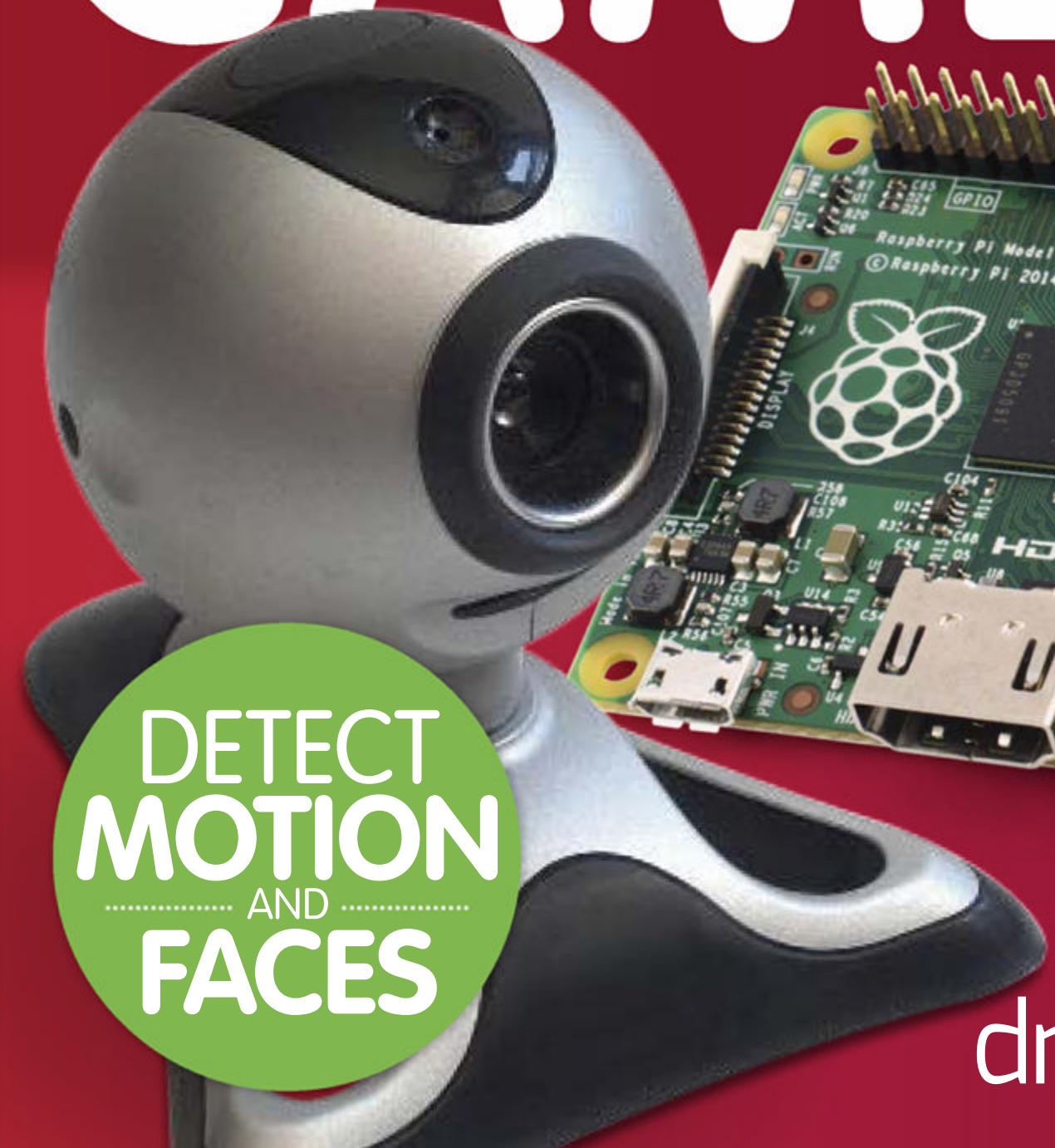
RasPi

DESIGN
BUILD
CODE

11

Get hands-on with your Raspberry Pi

HACK YOUR CAMERA



DETECT **MOTION** AND **FACES**

Plus Code a drum machine

Welcome

Picamera is a fantastic library for the Raspberry Pi camera module, and we took a good look at it back in issue 6, but you can do so much more with your camera module – and even USB cameras, if you're controlling them with your Raspberry Pi. Did you know that you can set up a regular webcam to recognise faces, for example? Or that you can create a *Matrix*-style bullet time effect by chaining lots of camera modules together? We'll show you how it's done and more in this special camera tricks issue of **RasPi**. And if photography isn't your passion, don't worry – we've also got a great audio sampler project for you, plus guides to mining Bitcoins and volunteering CPU power. Enjoy!

Gavin Thomas

Editor

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

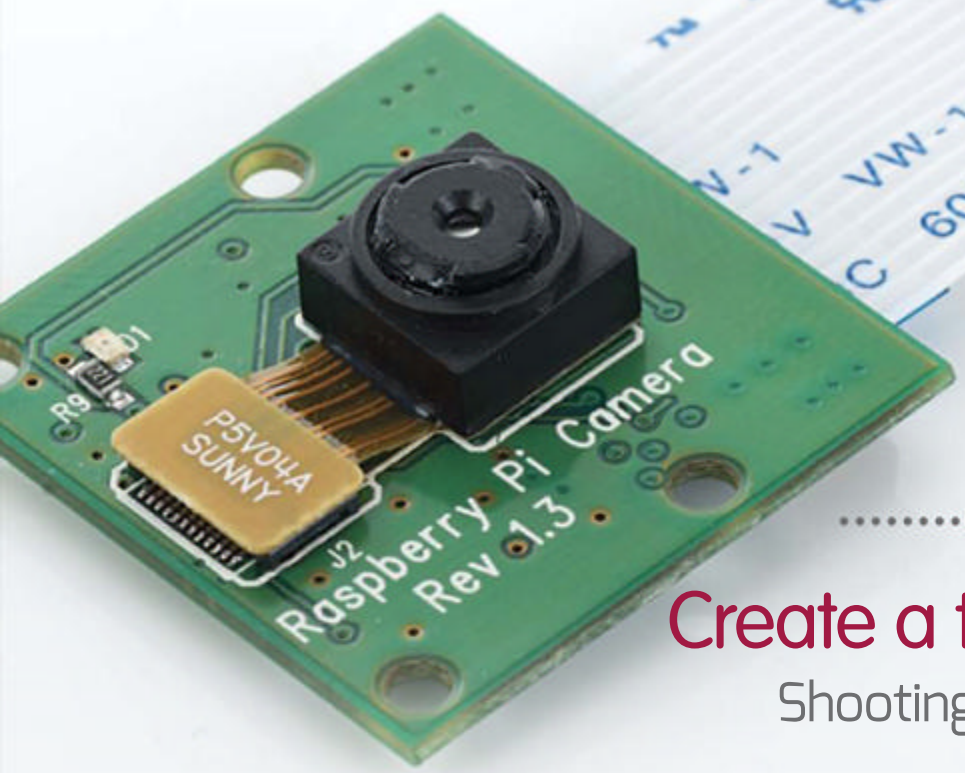
Learn to make and code gadgets with Raspberry Pi

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk



Contents

Create a time-lapse camera

Shooting time-lapse video with your DSLR camera is a cinch if you get Python and your Pi involved

You'd be forgiven for thinking that creating mesmerising time-lapse videos like those of Vincent Laforet (**www.laforetvisuals.com**) or John Eklund (**www.theartoftimelapse.com**) might be out of reach of the Average Joe, but with the help of the Raspberry Pi and a sprinkling of Python code, that's no longer the case. In this guide we're going to use our Raspberry Pi to trigger our run-of-the-mill DSLR camera (in this case a Canon EOS) to create pixel-perfect time-lapse imagery for little more than £10. Here's how…

THE PROJECT ESSENTIALS

Breadboard, connectors, jumper wire

DSLR camera

Compatible shutter cable

Raspbian with Python 2.7

01 Set up the Raspberry Pi

For this tutorial we're assuming you're using a recent build of Raspbian, which you can download from **www.raspberrypi.org/downloads**. With the Raspberry Pi set up with a keyboard, mouse and monitor, open the terminal and type:

```
$ sudo apt-get update
```

02 Install the RPi.GPIO library

Next we want to make sure your development environment is set up. Users of Raspian should be set up already, but you should follow these steps anyway to make sure. In the terminal, type:

```
$ sudo apt-get install python-dev
$ sudo apt-get install
python-rpi.gpio
```

03 Set up the Pi Cobbler

With the development environment set up, we can turn our attention to the hardware. For this tutorial we've used a cheap prototyping breadboard and an Adafruit

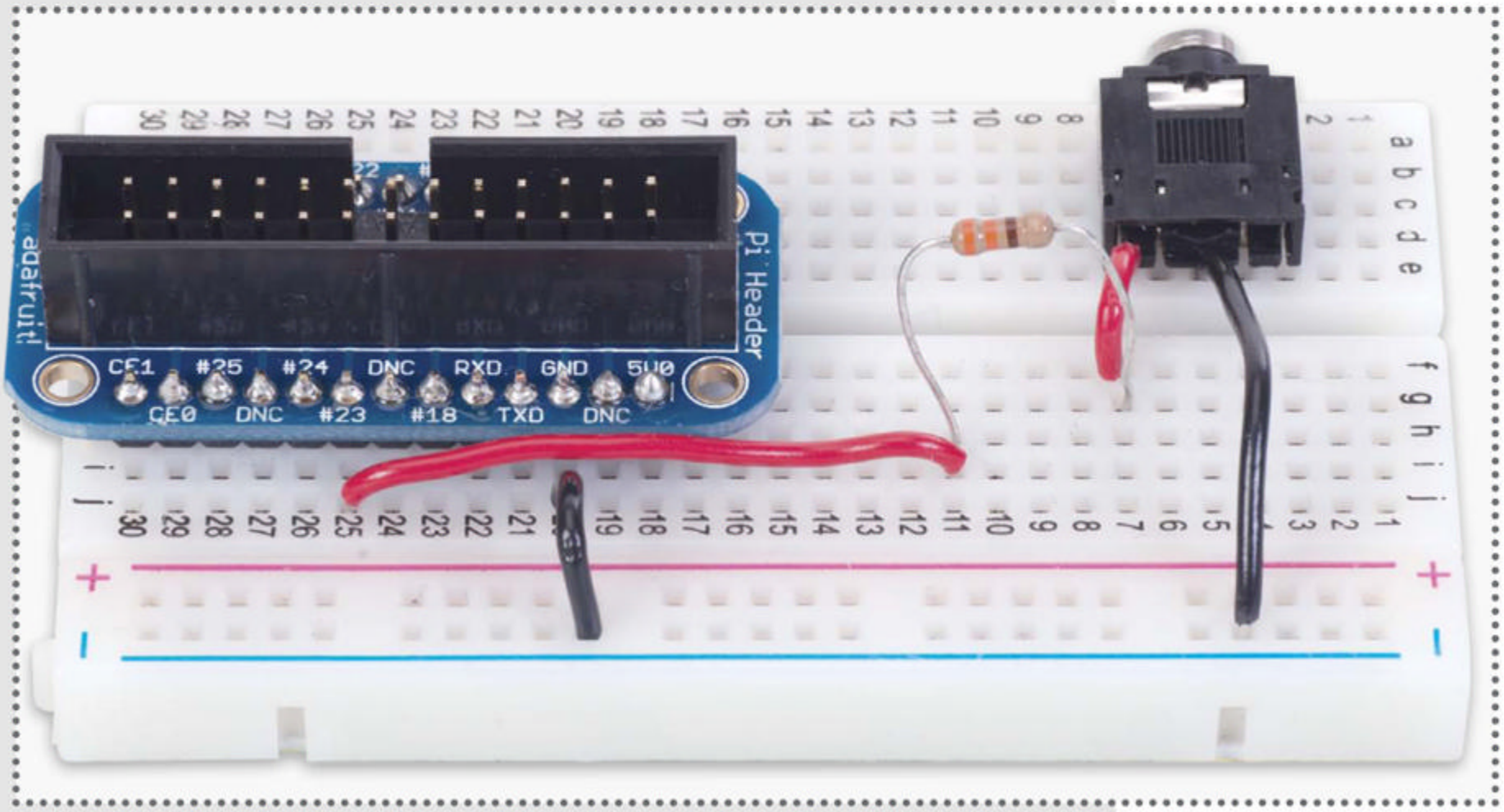
> "We're going to use our Raspberry Pi to trigger our run-of-the-mill DSLR camera to create pixel-perfect time-lapse imagery"

Pi Cobbler (**http://bit.ly/1LlRyS4**) to give us easy access to the Raspberry Pi's GPIO pins. As you can see from the picture at the bottom of this page, the Cobbler straddles the centre-point of the breadboard and a ribbon cable connects the two.

04 Configure the breadboard

For the Raspberry Pi's GPIO to control the camera, we need to create a circuit between a pin on the GPIO (in this case pin 23 on the Cobbler – but it's actually physical pin 16) and the pin that connects to the 'head' or 'tip' of the camera cable that activates the shutter when connected. We've also used a resistor, but it isn't required. The base of the connector cable is always ground, so make sure you ground the 'GND' pin on the Cobbler and the middle pin on the audio jack. It's very simple stuff. With the circuit complete, we can focus on the code.

Below Using a basic 3.5mm audio jack, we can pulse the shutter trigger simply by applying voltage to the cable 'head'



05 The RasPi Time-lapse Photography Tool

We've created a small 55-line Python utility called The Linux User Time-lapse Photography Tool, which asks the user to input how many shots they'd like to take and the frequency they'd like them taken – the full code listing begins on the next page. It then takes that information and uses it in a For loop to activate the shutter using GPIO pin 16 . If you'd like to use the project 'in the field' then we'd recommend using the Android app ConnectBot to SSH into your Ras Pi for input and feedback. Don't forget to start your script:

```
$ sudo python time_lapse_camera.py
```

> "If you'd like to use the project 'in the field' then we'd recommend using the Android app ConnectBot to SSH into your Ras Pi"

06 Creating a video from the images

With your camera packed with images, we need to now collect and output them as a video file. While it's technically possible to do on the Pi, we'd recommend copying them to an easily accessible folder on a separate Linux PC since there will be quite a lot of heavy lifting involved. We're going to use FFmpeg. With the terminal open in the folder where your images are stored, type:

```
$ ffmpeg -f image2 -i image%04d.jpg -vcodec
libx264 -b 800k video.avi
```

This obviously assumes you have libx264 installed on your machine and the 'image%o4d.jpg' assumes the file format and the number of digits it's dealing with (in this case: 'picture0001.jpg'). The flags we've selected should make for good results, but for full documentation of FFmpeg's incredible capabilities see: **http://ffmpeg.org/ffmpeg.html**.

The Code TIME-LAPSE CAMERA

```
import RPi.GPIO as GPIO
import time

print 'Welcome to the RasPi Time-lapse Photography Tool.'
print "Just tell us how many shots you'd like to take and the interval between them."
print "Try googling 'time-lapse interval calc' if you need help deciding."

def main():
    shots = raw_input('How many shots would you like to take? ->')
    interval = raw_input('How frequently do you want to take them (in seconds)? ->')

    if shots.isdigit() and interval.isdigit():
        shots = int(shots)
        interval = int(interval)

        print "You'll be shooting for %d minutes." % (shots * interval / 60)
        answer = raw_input('Are you ready to proceed?(yes/no):')
        confirm = answer.lower() in ['yes', 'y']

        if confirm:
            GPIO.setmode(GPIO.BOARD)
            GPIO.setup(16, GPIO.OUT)
            taken = 1
            print
            print 'Starting a run of %d shots' % (shots)

            for i in range(0, shots):
                print
                print 'Shot %d of %d' % (taken, shots)
                taken +=1
                GPIO.output(16, GPIO.HIGH)
                time.sleep(0.5)
                GPIO.output(16, GPIO.LOW)
```

The Code TIME-LAPSE CAMERA

```
                time.sleep(interval)
            GPIO.cleanup()
        else:
            print "Let's try again (or press Ctrl + C to quit):"
            main()
    else:
        print "Oops - You can only enter numbers. Let's try again:"
        main()

    print
    print 'Done. Thanks for using the RasPi Time-lapse Photography Tool!'
    again = raw_input('Would you like to do another time-lapse? (yes/no): -> ')
    proceed = again.lower() in ['yes', 'y']

    if proceed:
        main()
    else:
        print 'See you next time!'
        quit()

if __name__ == '__main__':
    main()
```

Detect motion with picamera

Use your camera module to detect motion and take a photo of the cause for home security

"Having the camera take photos when motion occurs reduces the strain on the Raspberry Pi"

The Raspberry Pi camera is an excellent piece of kit for many projects. With a decent sensor, video capabilities and great Python support from picamera, there's plenty you can do with it. Home surveillance setups and time-lapse photography with the camera module itself are both brilliant projects – but what if you could combine the two? Having the camera take photos when motion occurs reduces the strain on the Raspberry Pi for surveillance and can create a better time-lapse for more human situations. The code is fairly simple as well, plus you'll be able to learn some special methods of storing photos.

THE PROJECT ESSENTIALS

Latest Raspbian image

picamera Python module

Pi camera

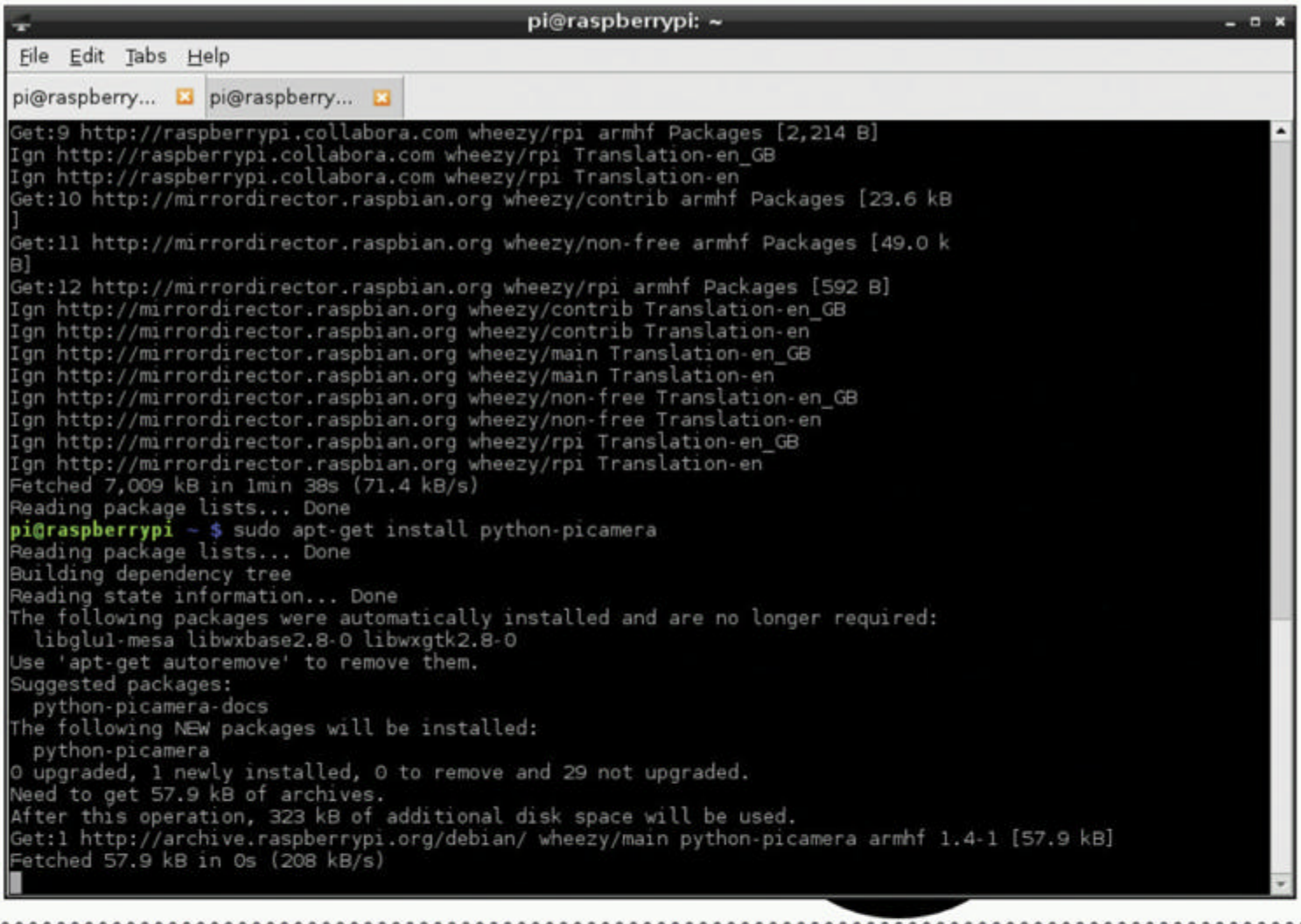
01 Enable the camera module

Before we start, make sure to update your Raspberry Pi and its firmware with **apt-get update** and **apt-get upgrade** followed by **rpi-update**. Once that's done, enter the **raspi-config** by typing exactly that in the terminal and choosing 'Enable' in the Enable Camera option. Hit Finish to reboot.

02 Attach the camera

Turn the Raspberry Pi off either during the reboot process or afterwards. Disconnect the power and find the slim port between the ethernet port and the HDMI port. Lift up the fastener and slot in the camera module's ribbon, with the silver side facing towards the HDMI port.

Below The camera adds about 200-250mA to the Pi's power requirements, so make sure you have a good supply



03 Install the Python modules

The code requires a couple of extra Python modules to work. First, the all-important picamera so we can better control the camera, along with the Python Image Library so we can inspect the images. Install them both with:

```
$ sudo apt-get install python-picamera
python-imaging-tk
```

04 Position your camera

Where are you placing your camera? Does the Raspberry Pi have adequate access to power? Are you controlling it with SSH or will it be in range of a display and keyboard/mouse? Set up your Raspberry Pi accordingly and take a sample picture to make sure it's in the right location with:

```
$ raspistill -o test.jpg
```

Above If you run into problems, you may have to try updating your firmware with **sudo rpi-update**

05 Tweak the sensitivity

In our code, difference and pixels are used to determine when a picture should be taken. The difference variable is the amount a pixel needs to change in colour to count as a change, while pixels is the number of these changed pixels that will be used to determine whether or not enough motion has been created to take a picture.

06 Cross the stream

To make the comparison, we're taking a sample image and inputting it into a stream we've created using io.BytesIO. It's stored in memory and then compared with the previous image that has been saved later on in the code – this determines whether or not a new photo should be taken.

Left With modular components, It's easy to disassemble and allows for various configurations to suit your needs

"Difference and pixels are used to determine when a picture should be taken. The difference variable is the amount a pixel needs to change in colour to count as a change"

The Code MOTION DETECTION

```
import io
import os
import picamera
import time
from datetime import datetime
from PIL import Image

camera = picamera.PiCamera()

difference = 20
pixels = 100

width = 1280
height = 960

def compare():
    camera.resolution = (100, 75)
    stream = io.BytesIO()
    camera.capture(stream, format = 'bmp')
    stream.seek(0)
    im = Image.open(stream)
    buffer = im.load()
    stream.close()
    return im, buffer

def newimage(width, height):
    time = datetime.now()
    filename = "motion-%04d%02d%02d-%02d%02d%02d.jpg" % (time.year, time.month,
time.day, time.hour, time.minute, time.second)
    camera.resolution = (width, height)
    camera.capture(filename)
    print "Captured %s" % filename
```

The Code MOTION DETECTION

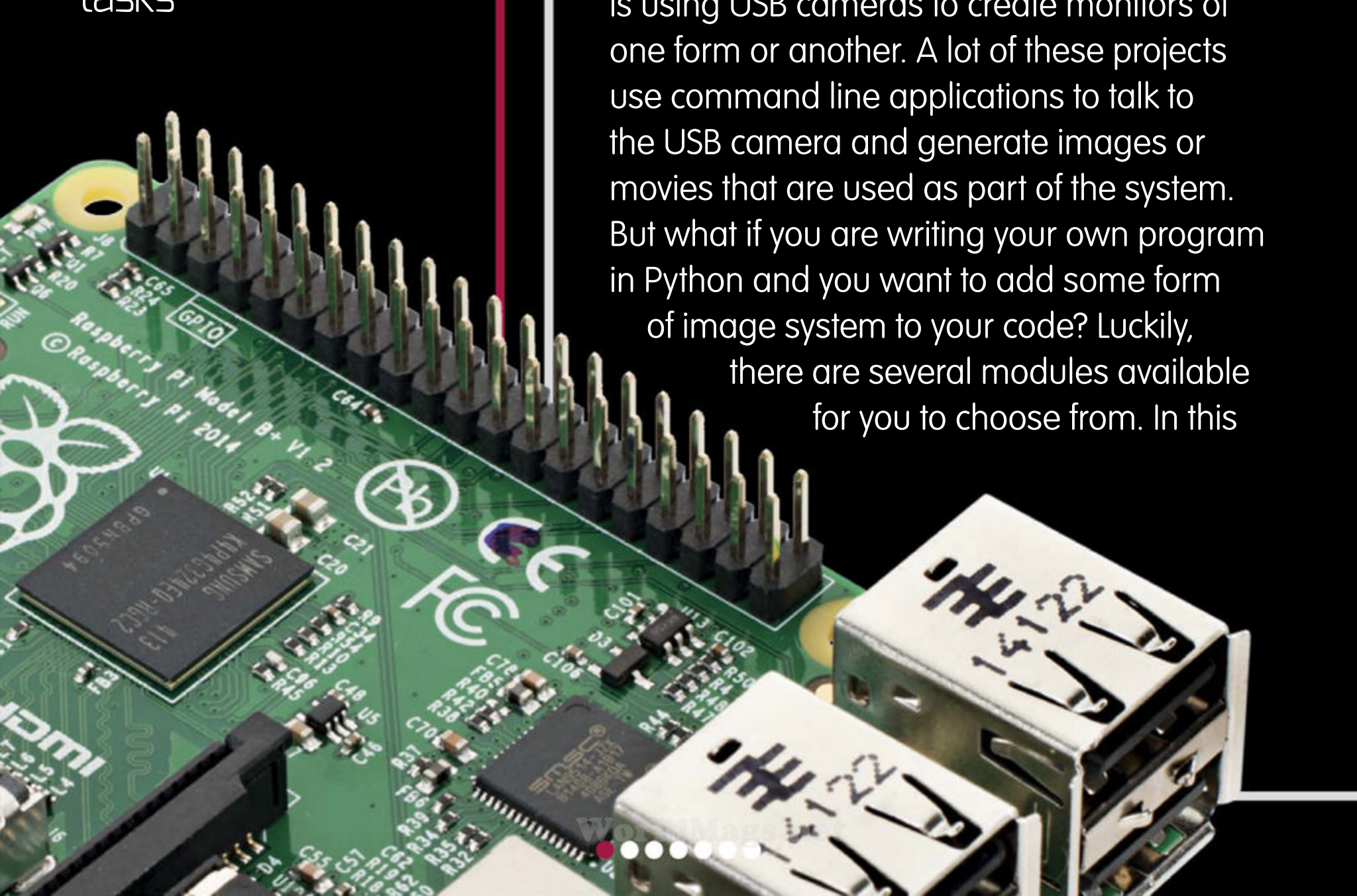
```
image1, buffer1 = compare()

timestamp = time.time()

while (True):

    image2, buffer2 = compare()

    changedpixels = 0
    for x in xrange(0, 100):
        for y in xrange(0, 75):
            pixdiff = abs(buffer1[x,y][1] - buffer2[x,y][1])
            if pixdiff > difference:
                changedpixels += 1

    if changedpixels > pixels:
        timestamp = time.time()
        newimage(width, height)

    image1 = image2
    buffer1 = buffer2
```

Turn your USB camera into a motion sensor

Learn how to implement facial recognition into your Raspberry Pi using Python and a webcam

"SimpleCV is built on top of OpenCV, making it easier to use for common tasks"

Back in issue 8, we looked at using the Kinect with the Raspberry Pi. But not everyone has access to this kind of hardware. Another class of project that is popular with Raspberry Pis is using USB cameras to create monitors of one form or another. A lot of these projects use command line applications to talk to the USB camera and generate images or movies that are used as part of the system. But what if you are writing your own program in Python and you want to add some form of image system to your code? Luckily, there are several modules available for you to choose from. In this



article, we will take a look at using SimpleCV to get your program to talk with the USB camera. SimpleCV is built on top of OpenCV, making it easier to use for common tasks. Assuming you are using Raspbian, you can go to the main page for SimpleCV (**www.simplecv.org**) and download a DEB file. To install it, you can simply run:

```
$ sudo dpkg -i SimpleCV-1.31.deb
```

Before you do, however, you will want to install all of the dependencies. You can do that with the command:

```
$ sudo apt-get install python python-support
python-numpy python-scipy ipython python-opencv
python-pygame python-setuptools
```

You can check that everything worked by running the command **simplecv** at the command line. This will start Python up and run the interactive shell that is provided by the SimpleCV module. You can then try connecting to your USB camera and pulling images from it.

Now that everything should be up and running, how do you actually use it in your own code? You can load all of the available functions and objects into the global scope with the command:

```
$ from SimpleCV import *
```

Making sure that you have your USB camera plugged in, you can now create a camera object with:

```
$ cam = Camera()
```

This will load the required drivers, and initialise the

camera so that it is ready to start taking pictures. Once this object creation returns, you can grab an image from the camera with:

```
$ img = cam.getImage()
```

At least in the beginning, when you are experimenting, you may want to see what this image looks like. You can do this with:

```
$ img.show()
```

You will, of course, need to have a GUI running in order to actually see the movie. Otherwise, you will get an error when you try and call **img.show()**. Don't forget that you can always pull up documentation with commands like:

```
$ help(cam)
$ help(img)
```

With the 'Image' object, you can do some basic processing tasks right away. You can scale an image by some percentage, say 90%, with **img.scale(90,90)**. You can also crop an image by giving it a start location and saying how many pixels across and how many up and down you want to crop to; this looks like **img.crop(100,100,50,50)**. SimpleCV has the location (0,0) as the top-left corner of an image.

The really interesting functionality in SimpleCV is the ability to find features within an image and to work with them. One of the clearest features you can look for is blobs, where blobs are defined as continuous

"One of the clearest features you can look for is blobs, where blobs are defined as continuous light regions"

light regions. The function **img.findBlobs()** will search the captured image for all blobs and return them as a FeatureSet. You can set the minimum number of pixels to consider a single blob, the maximum number of pixels, as well as a threshold value. If you are looking at a region that has some hard edges, you can use the function **img.findCorners()**. This function will return a FeatureSet of all of the corners within the captured image. A very simple monitor program could use one of these functions to see if there is any motion happening. If there is, the set of blobs or corners will change from one frame to another. Of course, a little more reading will lead you to the **img.findMotion()** function. This function takes two subsequent images and see if any motion can be detected going from one to the other. The default method is to use a block matching algorithm, but you can also use either the Lucas-Kanade method or the Horn-Schunck method.

The above methods will let you know some features of the captured images, and if any kind of motion has occurred. But what if you are more interested in identifying whether people have been moving around? Maybe you have an area you need to secure from espionage. In this case, you can use the function **img.findSkintoneBlobs()**. You can use a binarise filter threshold to set what constitutes a skin tone. If you need to do more, you have access to all of the underlying OpenCV functionality. One of these more advanced functions is face recognition. You can use the function **img.findHaarFeatures()** to look for a known type of object. If you wanted to look for faces, you could use something like:

```
$ faces = HaarCascade("./SimpleCV/Features/
HaarCascades/face.xml","myFaces")
```

```
$ img.findHaarFeatures(faces)
```

When you start developing these types of programs, one thing that might come into play is timing issues. You want to be sure that your code is fast enough to catch everyone that may be moving through the field of the camera. In order to figure out what is costing time, you need to be able to profile your code. The shell in SimpleCV provides a feature called **timeit** that will give you a quick and dirty profiling tool that you can use while you are experimenting with different algorithms. So, as an example, you can see how long the **findBlobs()** function takes on your Raspberry Pi with something like:

```
$ img = cam.getImage()
$ timeit img.findBlobs()
```

Once you find and fix the bottlenecks in your code, you can create the end product for your final version. With this article, you should now have enough to start using cameras from within your own programs. We have only been able to cover the bare essentials, however, so don't forget to go and check out the documentation covering all of the other functionality that is available inside the SimpleCV module.

> "You want to be sure that your code is fast enough to catch everyone that may be moving through the field of the camera"

The Code SIMPLECV

```
# SimpleCV provides a simple interface to OpenCV
# First, we will import everything into the local namespace

from SimpleCV import *

# Make sure your USB camera is plugged in,
# then you can create a camera object
cam = Camera()

# Getting an image from the camera is straightforward
img = cam.getImage()

# You can rescale this image to half its original size
img2 = img.scale(50,50)

# There are several features that you may want to look at

# You can extract a list of blobs
blobs = img.findBlobs()

# You can draw these blobs and see where they are on
# the image
blobs.draw()

# or a list of corners
corners = img.findCorners()

# If you want to identify motion, you will need two  # frames
img2 = cam.getImage()

# You can get a FeatureSet of motion vectors with
motion = img2.findMotion(img)
```

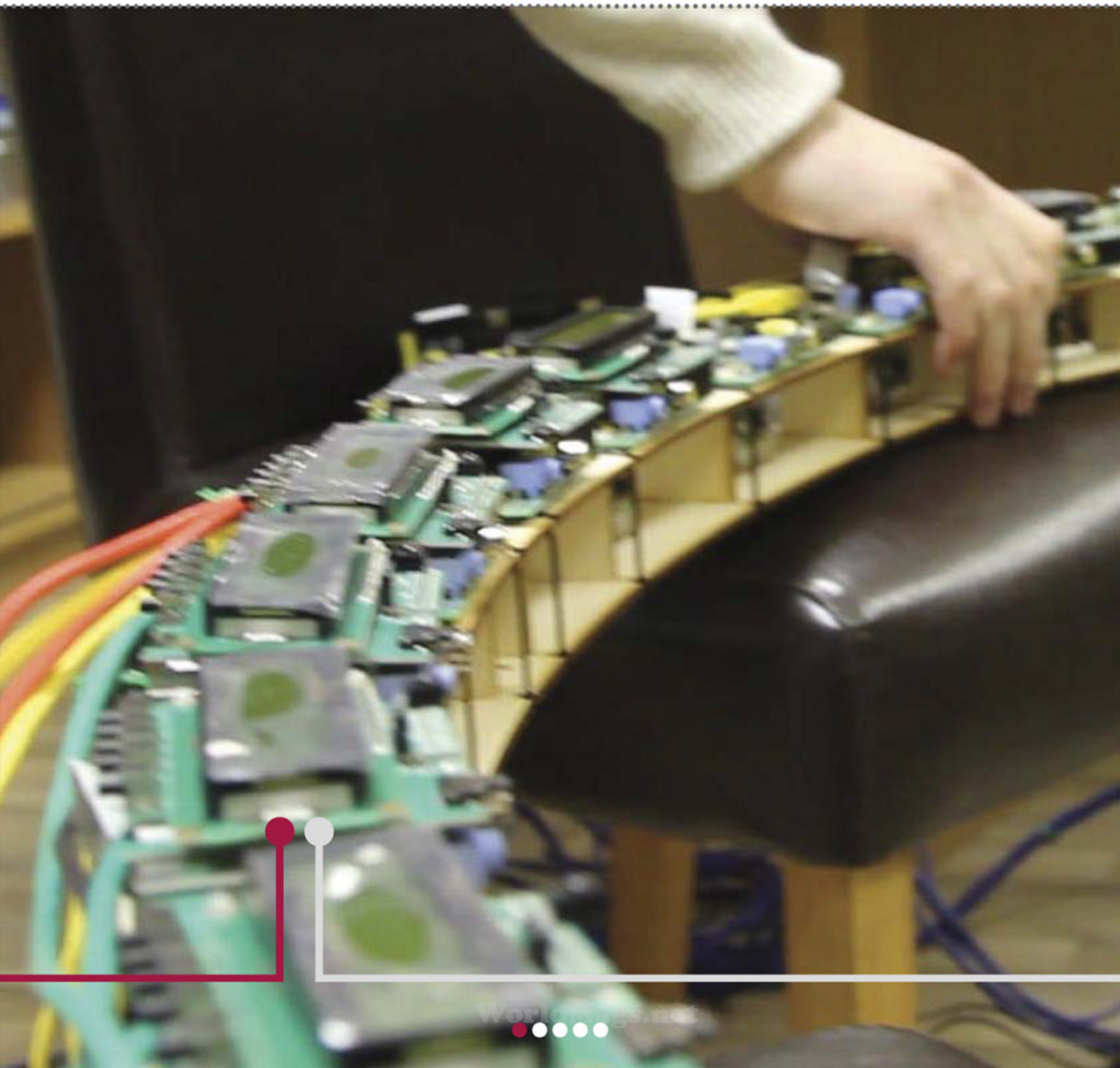
The Code SIMPLECV

```
# Face recognition is possible too. You can get a list of
# the types of features you can look for with
img.listHaarFeatures()

# For faces, you can generate a Haar Cascade
faces = HaarCascade('face.xml')

# Now you can search for faces
found_faces = img.findHaarFeatures(faces)

# You can load image files with the Image class
my_img = Image('my_image.jpg')

# You can save images to the hard drive, too
img.save('camera.png')
```

"You can use a binarise filter threshold to set what constitutes a skin tone. If you need to do more, you have access to all of the underlying OpenCV functionality"

Bullet Pi

PiFace creator Andrew Robinson shows us how you can freeze time by chaining 48 Raspberry Pi cameras



What was the inspiration behind Bullet Pi?

So I'd seen *The Matrix* and also a BBC programme called *Supernatural: The Unseen Powers Of Animals*, where they shot dolphins jumping out of the water and they could just freeze time, and then spin round so you saw the water drops, and I thought that was a really amazing effect. And reading how it was done, it was with a whole array of cameras and it seemed quite simple. So I had the Raspberry Pi and a couple of cameras sat on my desk, and it was sort of like, 'Well, if it seems that simple, then surely if we just got a whole load of Raspberry Pis and cameras – and they're not that expensive – why shouldn't we just be able to build it?' It was one of those moments where you think it should work but you don't actually know. So what I did was buy four cameras and I tried it, and that looked like it would work – it gave us a tantalising glimpse that we could do it.

So it was a good concept test?

Yeah, but even so it wasn't conclusive – it just told us that it's possibly likely but we still didn't actually know. So then it was just a case of buying a few more cameras. You get to the stage where all you can think is, 'Right, let's go for it. Let's have a full set of 48 cameras and just build it.' It was only really when we got the first set of images out, and then stitched that together, that you could see having a whole load of cameras and firing them at once is pretty much all we needed. It was one of those things where 'Wouldn't it be cool if…?' became 'You know, we might be able to…' and then 'Oh, we've done this!'

That's where the best Pi projects come from! So how do you set it off?

It's gone through a few iterations and, as we've tried it,

Andrew Robinson is an honorary research fellow of the University of Manchester, and the creator of PiFace Control and Display (learn more at **www.piface.org.uk**)

we've made a few refinements. In the latest setup, we've got all the Pis booted into Raspbian and, on boot, they run a little Python script we've got. We've fitted them so every Pi's got a PiFace Control and Display module that you can have simple interactions with using buttons – it just drives a menu. So that lets us know that all the Pis have booted up, because if you've got 48 Pis and a network cable comes out, it's tricky to find out which one when they all look the same. So that was really just a luxury in terms of helping to debug, and it also means we can reconfigure the array if we wanted to do a lateral version.

What are you using to stitch the images?

That's FFmpeg. We have had to do some things with the alignment just so you get a smooth video out, because some of the cameras are at slightly different angles and if you don't then the picture tends to go up and down. So basically we take a set of images with each calibration and work out whether there's a positive or negative offset to each frame, and then when we stitch them together we effectively chop off the top and the bottom and just use the middle of the frame.

Is that handled by a script or do you do it manually once you've taken the shots?

We actually trigger it as two points, but there's nothing to stop the two scripts being put together in the command line and all being run at once. The only reason we run it twice is that the script to stitch the images together takes about a minute and sometimes, if you take an image and then the person you're taking the image of says 'I don't like that, let's do it again', it saves cancelling something that's already started. So for logical reasons we run it as two separate operations but effectively they're just two

If you like

If you're thinking of using PiFace Control and Display in one of your own projects, check out the excellent tutorials provided by the team over at: **piface.org.uk/guides**

Further reading

To see Bullet Pi in action, check out this video from its first outing at the Manchester Raspberry Jam in 2013: **bit.ly/1wloP6y**

commands: one command to take the images and one command to stitch the images, and there's nothing to stop it all being automated. You could put a pressure pad in there so that when people stood in the middle of the bullet rig it would automatically trigger.

Is the PiFace a crucial component or was it just something you wanted to use?

It was a luxury for us that we had it kicking around. It turned out to save us time when we were debugging. It's not essential – you can build it without – but it made life much easier. We've also tried it in other configurations, and when you put it in other configurations, just being able to go through a menu on each Pi and then set where it is in the sequence is a lot easier if you can actually relate to the Pi that you've got in your hand, rather than having to look at what number it is, SSH into it and set a file.

What configurations have you tried?

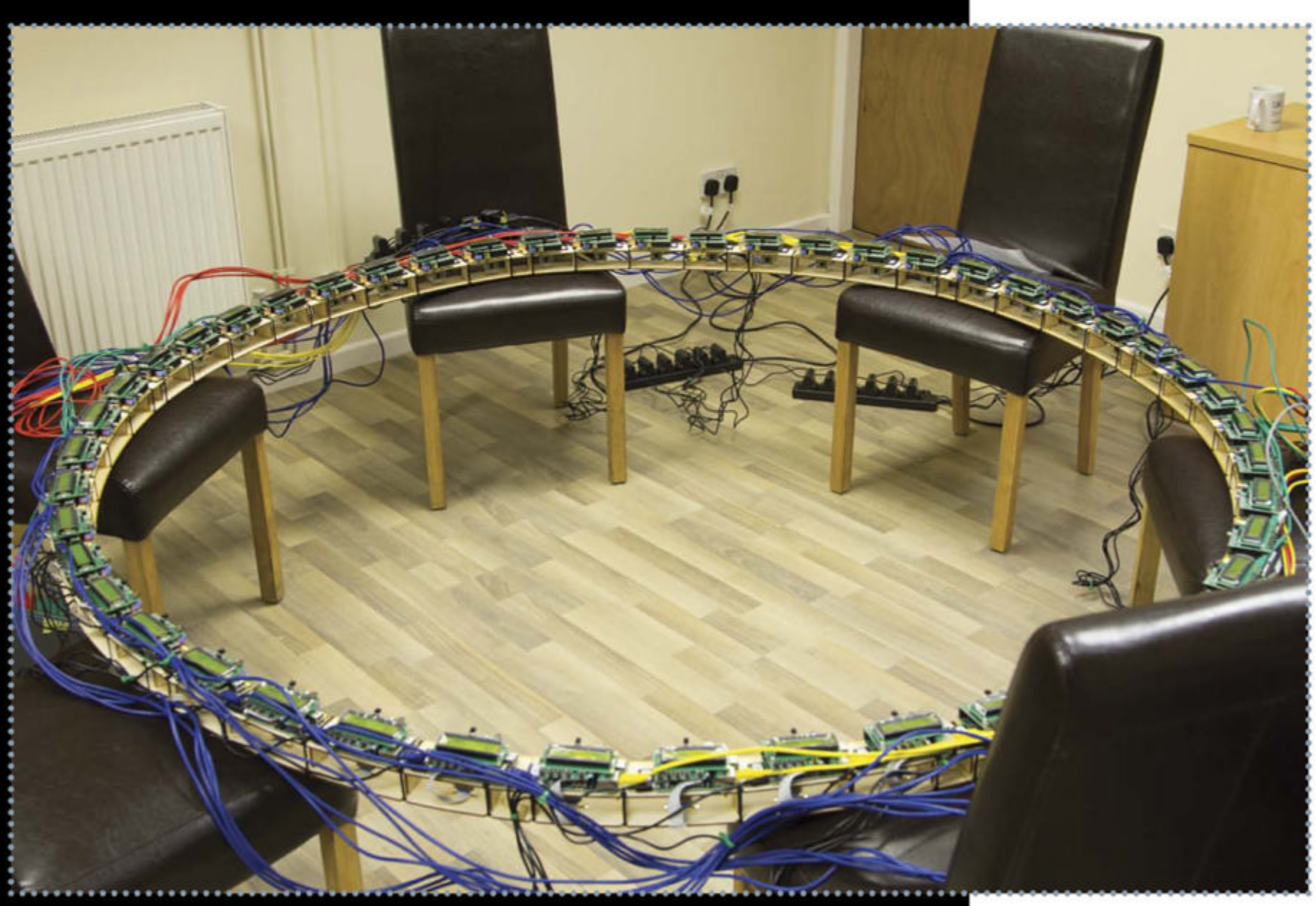
We tried semicircles and also we've tried ones in straight lines, and that way you get the effect of the camera panning along as opposed to spinning around a point. We've got future plans, which do involve taking it into more wild and wacky situations. So far we've learned from the examples we've done inside, and then as we've got the setup faster and more automated and got all the little tweaks and experiences from it, we're becoming more ambitious with it. There are definite plans to mount it on impressive things outside and get some sports photography going with it. There's one person who actually booked us for their wedding after they'd seen it – they wanted photographs of their guests and it would be different, something unusual to capture the guests in a 360-degree spin. That's the other nice thing about

it – community. You put something out there online and suddenly it goes off in a different direction.

Do you plan to take it to any more Jams?

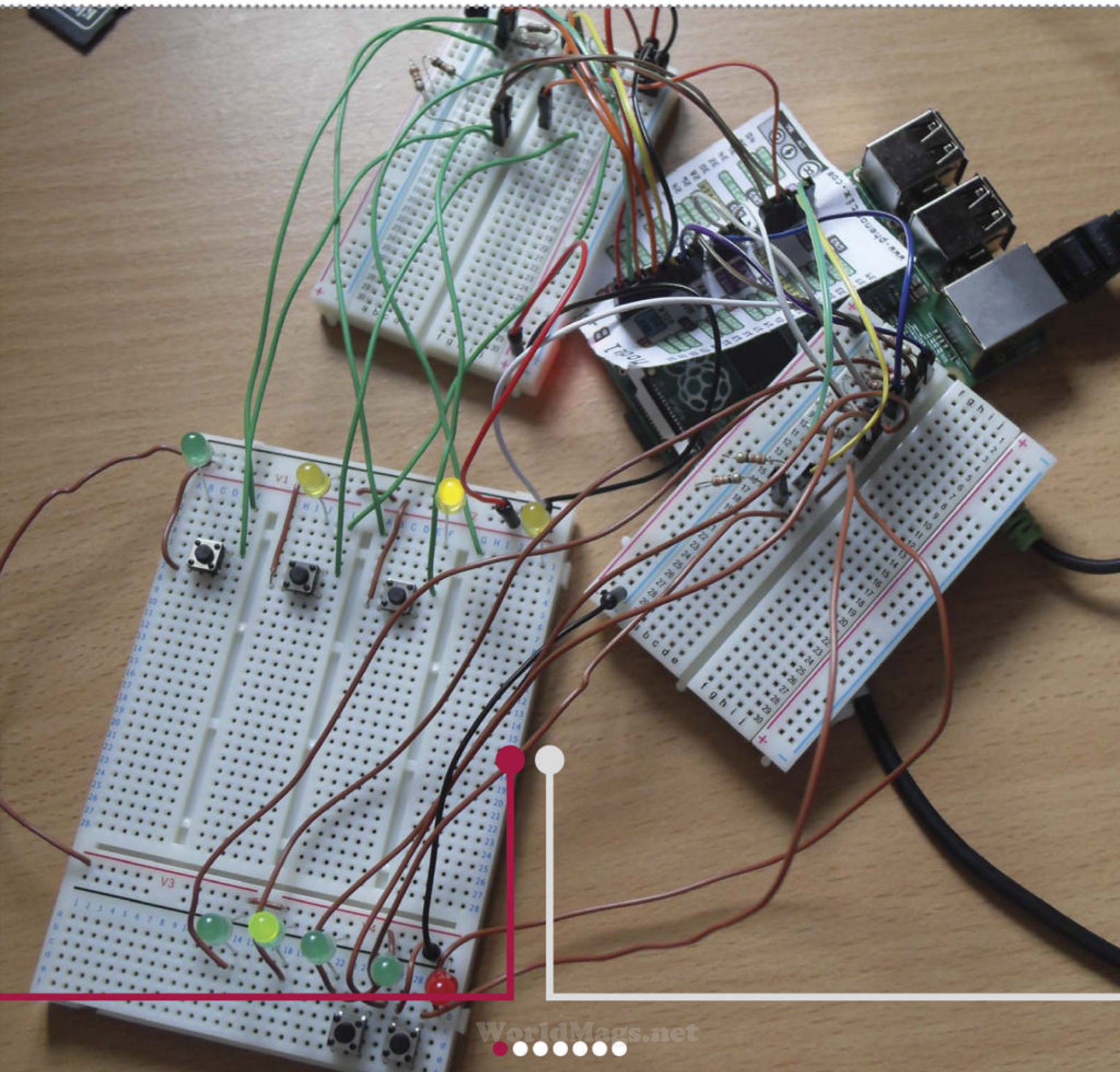
I think we'd like to do more things with it. One of the things I'd like to do is try and play around more with fusion of movements. So rather than having someone standing in the circle and then performing in the circle, see whether we could actually get a portable version and take the rig around – that way you could follow the action, if you like, and also get the video coming out of the Raspberry Pis. I would also like to experiment with background motion blur, things like that. We also think fireworks could be quite interesting, come November time.

Below Many people wondered if Andrew and the team were building a Large Hadron Collider when the first few photos were teased online



Make a drum machine

Build your own looping drum machine with only 200 lines of code!



Let's combine programming, electronics and music to create a simple sampler with looping capabilities. The implementation used in this article has three drum sounds as the samples, but it is trivial to add more until you run out of GPIO pins.

Before we start, we'll cover some terms. Music is split into bars. There are a certain number of beats in a bar. This sampler uses the 4/4 time signature, so there are 4 beats in each bar. Tempo is the speed at which music is played, and it is measured in beats per minute (bpm). A metronome is an audible tone that is heard at the start of every beat.

Quantization is the process of aligning notes to beats, or exact fractions of a beat, and a quantization value is usually given in the form 1/8. This means that there are eight possible places in a bar where a note can be played. When the sampler is recording and a sample button is pressed, we store the sample at the current position in the bar with the accuracy of the quantize value. There's a lot to cover, so let's get started.

THE PROJECT ESSENTIALS

Latest Raspbian image

Breadboard(s)

Push buttons

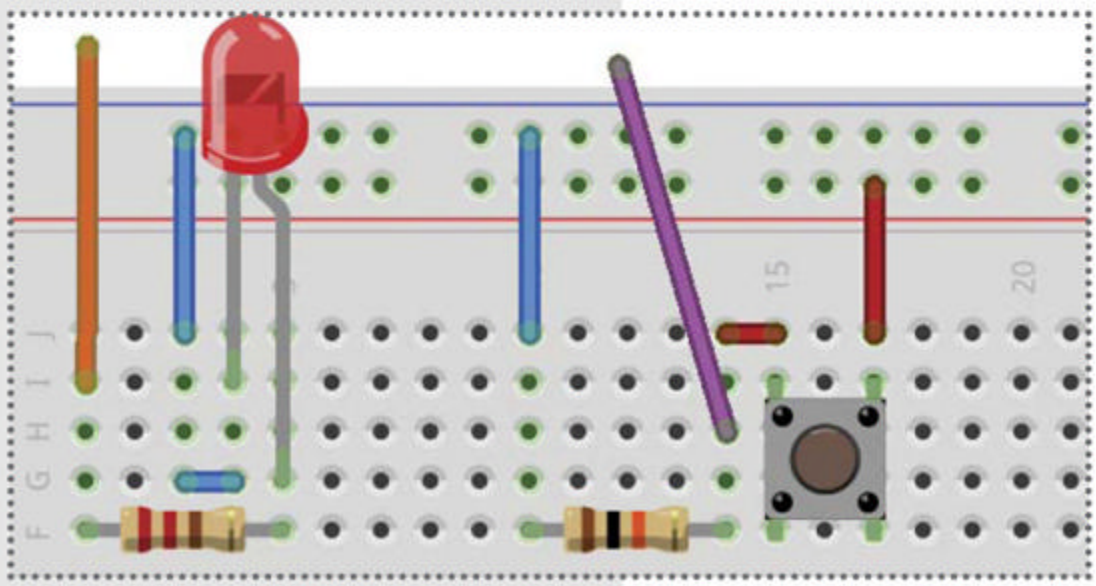
LEDs

Female-to-male GPIO jumper cables

Male-to-male GPIO jumper cables

01 Connect LEDs

The circuit diagram is an LED that can be turned on and off with a GPIO output. The orange wire is the connection from the GPIO output. This then goes through a 220Ω resistor to limit the current draw to a safe level. This current flows through the positive leg of the LED and then back to ground. We need nine LEDs for this project.



02 Wire up buttons

The second circuit we need is a push button. The purple wire goes to a GPIO input. There is a 10KΩ pull down resistor to ground, which represents a logical 0. When the push button is pressed, the 3.3V supply representing a logical 1 is connected to the purple wire. The electricity takes this path because it has less resistance than the path to ground. We need two buttons for record and undo, and then as many buttons as you like for samples (three drum samples are provided).

03 Download samples

Create a few folder for the project called pisampler. Then download and unzip the sounds:

```
$ mkdir pisampler
$ cd pisampler
$ wget http://liamfraser.co.uk/lud/pisampler/sounds.zip
$ unzip sounds.zip
```

There will now be a folder called sounds with some samples in. The file format for samples is .wav audio, Microsoft PCM, 16 bit, stereo 44100 Hz. Mono will work too. Any samples can be converted to this format with Audacity by exporting them as a .wav file.

04 Import required libraries

Create a file called pisampler.py. We need to import the required libraries and set configuration values – go to the annotated code at the end of this guide and input the Step 4 section. A key option is debounce: the time to wait before a button can be pressed again to stop accidental presses from contact bounce.

"We need two buttons for record and undo, and then as many buttons as you like for samples (three drum samples are provided)"

05 Create a sample class

We're going to use a class to represent each sample. It's going to need a few things: the pin that the sample button is connected to, the name of the sound file, and a reference to the instance of the sampler class. We haven't created the sampler class yet, but the sample will need to be able to tell if the sampler is recording or not, and have access to the data structure that recordings are stored in to add itself to it if necessary. Again, check the annotated code.

The other thing that we need to do is set the GPIO pin to an input, and add an event listener for when the button is pressed. We set callback (the function to be executed when the button is pressed) to a function called self.play_btn, which will play a sound and add it to the recording data if we are recording. It will become clear how this works once we've written the sampler class. Note that the GPIO event handler passes the pin that the event handler was triggered on, hence the channel variable that is present but never used.

06 The sampler init method

Here's the start of the sampler class. The last value in the Pygame mixer init is the buffer size. You might need to increase this to 1024 or higher if you have audio dropouts. We create some variables to store recording state. Metronome sounds are then added and their volume lowered. We also create a list to hold our samples in. We create nested arrays to represent recorded sample presses. There is an array for each bar. Each bar has an array for each possible quantize value. The default value of 64 gives us 64 possible places to store a sample hit per bar.

"The last value in the Pygame mixer init is the buffer size. You might need to increase this to 1024 or higher if you have audio dropouts"

Finally, we set up the LED pins, and the pins for the record and undo buttons.

07 The tempo property

The tempo variable is actually a property with a custom setter. This means when a value is assigned, it does a custom action. In our case, we need to calculate how often we need to check for recorded notes to play in the main loop that we'll write later.

08 Helper functions

There are a few helper functions in the class. One of them simply adds a sample to the list of samples. Another sets a variable to trigger recording at the start of the next loop. There is also a function which turns the red LED on when the recording variable is set to true. Now we'll jump forward and take care of the main loop towards the end of the full code listing.

"There is also a function which turns the red LED on when the recording variable is set to true"

09 Start the main loop

The main loop doesn't actually have to do any work at all to play sounds, as that's done by the GPIO event handlers. The main loop is used to play the metronome, update the state about which bar/beat/quantize we are currently on, update the LEDs and deal with recording if necessary.

Before the loop, we create variables to track the state. The last recorded loop is a list that we will use as a stack. A stack is a last in/first out data structure, allowing us to undo recordings multiple times by removing each sample that was recorded when the loop count was the value on the top of the stack.

If we're at the start of a new beat then we use a function called do_leds that we haven't created yet. As

the LEDs work in the same way (a block of four LEDs where only one is turned on), we can use the same function twice and just pass a different set of pins, and the index of the LED we want to turn on. We then call the do_metronome function which will play the appropriate metronome sound.

We then do some recording logic which starts recording if we should be recording, and stops recording if we have just been recording, adding the loop number to the last_recorded_loop stack. We increment the loop count after doing this.

10 Main loop continued

This code is at the indentation level after the "while True:" statement. After dealing with the recording logic, we need to play any notes that have been previously recorded. We'll work out how to do that later on. After that, we can sleep until the next quantize change is due. Once this happens, we have to do logic that deals with the quantize and any related variables such as the beat or bar if necessary, either incrementing them or resetting them if necessary.

11 Lighting LEDs

The LED simply goes through each pin in the list you provide it with and lights up the appropriate LED, ensuring that all of the others are turned off.

12 The metronome

The metronome simply plays a high tone on the first beat or a lower tone on the remaining beats if the metronome variable is set to true.

"After dealing with the recording logic, we need to play any notes that have been previously recorded"

13 The recording code

Looking at the sample class we first created, you can see that if recording is enabled and a note is pressed, we add a dictionary to the list of samples for the current bar at the current quantize point. The dictionary contains a reference to the sample so that it can be played, and also the loop that it was added on so that it can be removed if necessary. The code for playing and undoing recordings can be seen below. Note that we directly play the sound rather than using the btn_play function so that we don't trigger the recording logic when playing recorded sounds. The pop function in undo_previous_loop removes the last thing that was added to the stack, which will be the loop count. We then go through every possible recording data point and remove anything recorded on the loop to be removed.

> "We directly play the sound rather than using the btn_play function so that we don't trigger the recording logic when playing recorded sounds"

14 Finishing it off

Now we need to add a main function where we load some samples in and start the main loop. Remember to run the code with **sudo python2 pisampler.py**, as we need sudo to access the GPIO. Happy jamming!

15 Possible improvements

There are a number of improvements that could be made to the sampler. Here are a few ideas:

- A button to turn the metronome on and off
- The ability to time stretch samples (such as chords) to fit with the tempo
- The ability to pitch shift samples on the fly
- Using a shift register to use less pins when lighting the LEDs, allowing more inputs
- The ability to save recorded beats for playback

The Code PI SAMPLER

04

```
import RPi.GPIO as GPIO
import time
import pygame
import os

beat_leds = [2, 3, 4, 17]
bar_leds = [27, 22, 10, 9]
record_led = 11
record = 19
undo = 26
debounce = 200 # ms
```

05

```
class Sample(object):
    def __init__(self, pin, sound, sampler):
        self.sampler = sampler
        self.name = sound
        self.sound = pygame.mixer.Sound(os.path.join('sounds', sound))
        self.pin = pin
        GPIO.setup(pin, GPIO.IN)
        GPIO.add_event_detect(self.pin, GPIO.RISING, callback=self.play_btn,
                              bouncetime=debounce)

    def play_btn(self, channel):
        self.sound.play()
        s = self.sampler
        if s.recording:
            s.recording_data[s.bar_n][s.quantize_n].append({'loop' : s.loop_count,
                                                            'sample' : self})
```

06

```
class PiSampler(object):
    def __init__(self, tempo=80, quantize=64):
        pygame.mixer.pre_init(44100, -16, 1, 512)
        pygame.init()
```

06

```
        self.quantize = quantize
        self.tempo = tempo
        self.recording = False
        self.record_next = False

        self.metronome = False
        self.met_low = pygame.mixer.Sound(os.path.join('sounds', 'met_low.wav'))
        self.met_high = pygame.mixer.Sound(os.path.join('sounds', 'met_high.wav'))
        self.met_low.set_volume(0.4)
        self.met_high.set_volume(0.4)

        self.samples = []
        self.recording_data = []
        for i in range(0, 4):
            bar_arr = []
            for i in range(0, quantize):
                bar_arr.append([])

            self.recording_data.append(bar_arr)

    GPIO.setmode(GPIO.BCM)
    for pin in beat_leds + bar_leds + [record_led]:
        GPIO.setup(pin, GPIO.OUT)

    GPIO.setup(record, GPIO.IN)
    GPIO.add_event_detect(record, GPIO.RISING,
                          callback=self.record_next_loop,
                          bouncetime=debounce)
    GPIO.setup(undo, GPIO.IN)
    GPIO.add_event_detect(undo, GPIO.RISING,
                          callback=self.undo_previous_loop,
                          bouncetime=debounce)
```

```
    @property
    def tempo(self):
        return self._tempo

    @tempo.setter
    def tempo(self, tempo):
        self._tempo = tempo
        self.seconds_per_beat = 60.0 / tempo

        self.quantize_per_beat = self.quantize / 4
        self.quantize_seconds = self.seconds_per_beat / self.quantize_per_beat
```

07

```
    def add(self, sample):
        self.samples.append(sample)

    @property
    def recording(self):
        return self._recording

    @recording.setter
    def recording(self, value):
        self._recording = value
        GPIO.output(record_led, value)

    def record_next_loop(self, channel):
        self.record_next = True
```

08

```
    def play_recording(self):
        for sample_dict in self.recording_data[self.bar_n][self.quantize_n]:
            if sample_dict['loop'] != self.loop_count:
                sample_dict['sample'].sound.play()

    def undo_previous_loop(self, channel):
```

13

13

```
        if len(self.last_recorded_loop) == 0:
            print "No previous loop to undo"
            return

        print "Undoing previous loop"

        loop = self.last_recorded_loop.pop()

        for bar in self.recording_data:
            for quantize in bar:
                removes = []
                for sample in quantize:
                    if sample['loop'] == loop:
                        removes.append(sample)

                for sample in removes:
                    quantize.remove(sample)
```

11

```
    def do_leds(self, leds, n):
        count = 0
        for led in leds:
            if count == n:
                GPIO.output(led, True)
            else:
                GPIO.output(led, False)

            count += 1
```

12

```
    def do_metronome(self):
        if not self.metronome:
            return

        if self.beat_n == 0:
```

12
```
            self.met_high.play()
        else:
            self.met_low.play()
```

09
```
    def run(self):
        self.loop_count = 0
        self.last_recorded_loop = []
        self.bar_n = 0
        self.beat_n = 0
        self.quantize_beat_n = 0
        self.quantize_n = 0

        while True:
            if self.quantize_beat_n == 0:
                self.do_leds(beat_leds, self.beat_n)
                self.do_leds(bar_leds, self.bar_n)
                self.do_metronome()

                if self.quantize_n == 0 and self.bar_n == 0:
                    if self.record_next:
                        self.recording = True
                        self.record_next = False
                    elif self.recording:
                        self.recording = False
                        self.last_recorded_loop.append(self.loop_count)

                    self.loop_count += 1
```

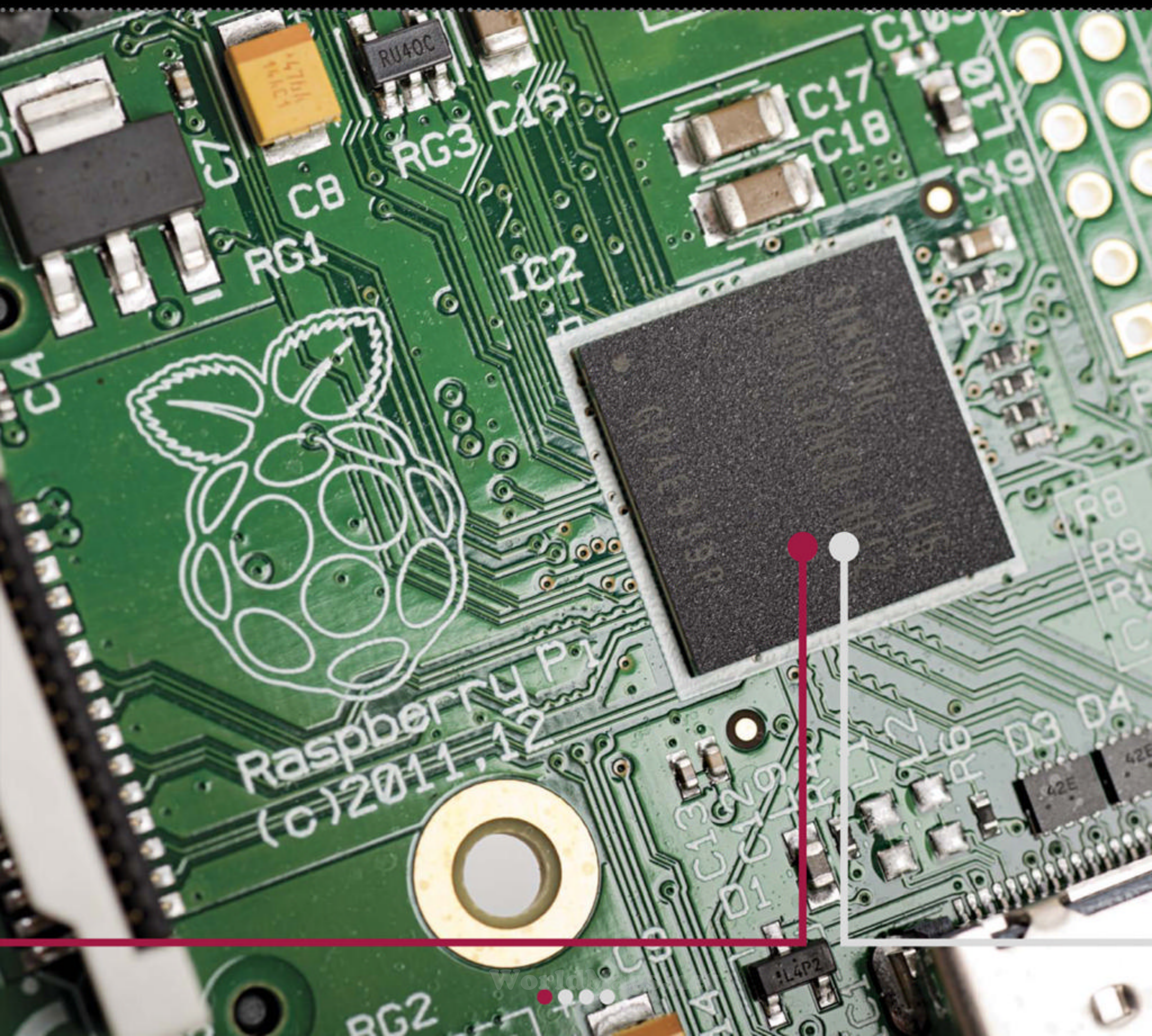
10
```
            self.play_recording()
            time.sleep(self.quantize_seconds)

            if self.quantize_beat_n == self.quantize_per_beat - 1:
                self.quantize_beat_n = 0
```

```
                self.beat_n += 1
            else:
                self.quantize_beat_n += 1

            if self.quantize_n == self.quantize - 1:
                self.quantize_n = 0
            else:
                self.quantize_n += 1

            if self.beat_n == 4:
                self.beat_n = 0
                self.bar_n += 1
                if self.bar_n == 4:
                    self.bar_n = 0
```

```
if __name__ == "__main__":
    sampler = PiSampler(tempo=140)
    sampler.add(Sample(05, 'kick01.wav', sampler))
    sampler.add(Sample(06, 'snare01.wav', sampler))
    sampler.add(Sample(13, 'clhat01.wav', sampler))
    sampler.metronome = True
    sampler.run()
```

What is ARM?

The chip powering the Raspberry Pi is known as an ARM chip. Here we explain exactly what it is and how it contributes to the Pi



Q You know, I've never really thought about it before, but the Raspberry Pi runs on an ARM chip, right?
A That's right, the Broadcom BCM2835 or BCM2836.

Q Okay, but what exactly is ARM?
A ARM is a type of CPU; the thing that does all the calculating and processing and is generally considered to be the 'brain' of the computer, as our teachers used to say in school.

Q So it's a type of CPU – what does that actually mean, though?
A Well, it's a CPU architecture. This means that it uses specific types of commands when programs need to interact with the system. This includes how it handles the memory, addresses data and all the really low-level stuff you'll rarely ever get to see.

Q ARM is a type of architecture – does that mean there are other kinds?
A Absolutely – in fact, there are many. Right now the most popular ones are ARM and x86 – the latter of which you may have heard mentioned before with regards to PCs.

Q What's the difference between x86, ARM and the other architectures?
A Technically they all work slightly differently, handling requests in a different manner. In a more practical sense, the main difference between the two is the applications for which they are used: x86 chips are commonly used in full systems such as laptops, desktop PCs and servers, whereas ARM is mainly used for mobile devices such as phones or embedded computers.

Back to school

BBC Micro
It all starts in the Eighties with the BBC Micro: a computer built by Acorn for the BBC's Computer Literacy Project, which created a generation of excellent British programmers. This included Raspberry Pi co-creator Eben Upton – the Micro was actually the inspiration for the Raspberry Pi.

Acorn computers
After the BBC Micros began to die out, Acorn continued to create the Acorn brand of computers using the ARM chips and RISC OS. These lasted throughout the Nineties before Acorn spun off to purely make ARM chips for mobile and embedded computers.

Q Why is this the case?
A It's usually attributed to power consumption – ARM chips are smaller and not generally as powerful, but they are also a lot more energy efficient. This makes them perfect for mobile phones and tablets.

Q So why don't laptops use ARM chips?
A Well, on the one hand, laptops are big enough to store larger batteries in them, but there's also the issue that operating systems for them are mostly made for x86 architectures. As laptops have always really been x86, it would create a lot of extra work for operating system developers and software developers.

Q Are there no ARM versions of major operating systems then?
A There are but they are less common and more recent. The Linux kernel supports ARM just fine, which is why you use a Linux distribution on Raspberry Pi. Android is also based on Linux and Windows 8 has an ARM version as well, although that's partly based on the mobile OS. There is an Ubuntu ARM version but it doesn't support the original Raspberry Pi.

Q Wait, it's ARM but it doesn't support the Raspberry Pi's particular type of ARM processor?
A Well, there are different versions of ARM chipsets: for example, the old Raspberry Pi uses ARM v6, the Pi 2 uses ARM v7, while modern phones are up to ARM v8 and 64-bit ARM processors. Ubuntu requires ARM v7 to work and is not making an ARM v6-compatible version. Recently, Canonical released a version called Snappy Ubuntu Core that works just fine on the Raspberry Pi 2.

Back to school

Raspberry Pi
The latest computer for schools uses an ARM processor from the remnants of Acorn computers and references the BBC Micro in its mission and naming. It's also helping further ARM as a desktop processor and making it more open.

Q So Intel made x86 – who made ARM?

A The clue is actually in the acronym: Acorn RISC Machine. It was originally designed by the British computing company Acorn Computers to be used in its hardware. It was very popular in British schools during the Nineties thanks to a coupon program put on by a major supermarket every year: Acorn computers were some of the rewards you could get with the vouchers. RISC is the operating system that ran on Acorn computers, designed to work specifically on ARM.

Q Interesting! So ARM, which started off as a proper computer processor, became ubiquitous with mobile. Is it the same RISC that I can download from the Raspberry Pi website?

A Near enough, yes. There has obviously been some development over the past 15-20 years but it's the same operating system as the one you would have used if you were going to school in the UK in that time period. Middle-click and everything.

Q So a lot of the Raspberry Pi is linked back to British school computing?

A From a certain perspective, yes, you could say that.

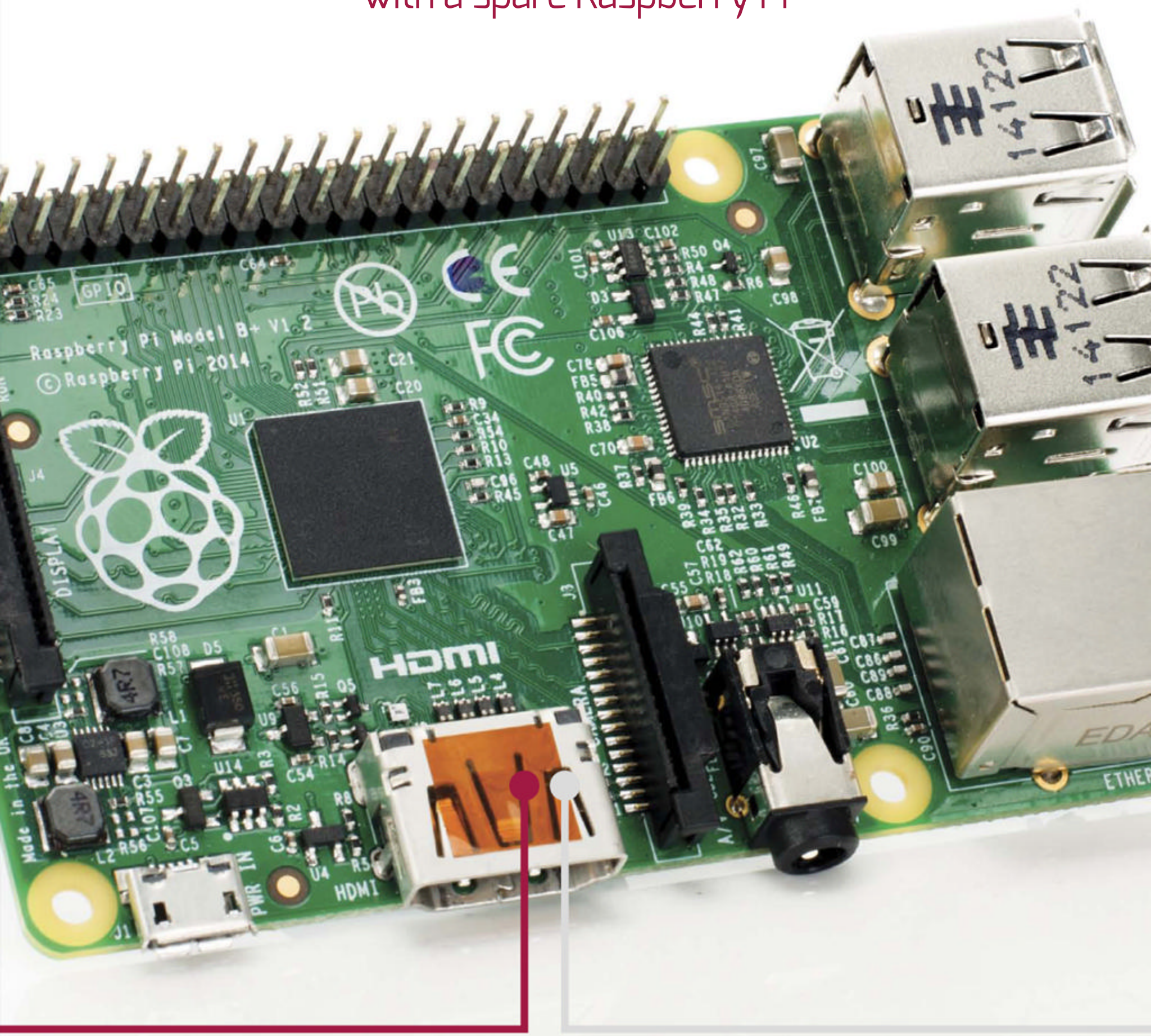
Q Which chip does the Raspberry Pi use specifically, then?

A It's the Broadcom BCM2835 in the original Pis and the newer BCM2836 in the Pi 2. It's actually a system-on-a-chip, so it's got CPU and graphics with native 1080p decoding on the chip. It's small, powerful and cheap, which makes it perfectly suited for the Raspberry Pi.

"RISC is the operating system that ran on Acorn computers, designed to work specifically on ARM"

Mine Bitcoins

Jump on the cryptocurrency bandwagon and mine some of your own Bitcoins – or other currency – with a spare Raspberry Pi



The concept of cryptocurrencies has come about in recent years as a sort of reaction to the way standard currencies are controlled. Cryptocurrenices such as Bitcoin are decentralised and not controlled by any one entity. In the past couple of years, Bitcoin has taken off to become a very valuable commodity, with whole Bitcoins worth hundreds of pounds. While you can trade your standard currency for a Bitcoin, you can also mine them with a computer.

The concept of mining basically means that you're increasing the security of the Bitcoin system by logging transactions properly; users who donate processing power to this endeavour get paid for it in Bitcoins. This is where we come in, turning your Raspberry Pi into a Bitcoin mine.

THE PROJECT ESSENTIALS

Latest Raspbian image

cpuminer

Bitcoin pool

01 Install dependencies

In this tutorial we're going to be using cpuminer to mine for Bitcoins; this needs to be compiled from source though, so we have to install some dependencies first to make sure it works. Do this with:

```
$ sudo apt-get install gcc gcc-4.5 g++ g++-4.5
libstdc++6-4.5-dev libpcre3-dev libcurl3-dev
make less
```

02 Download locations

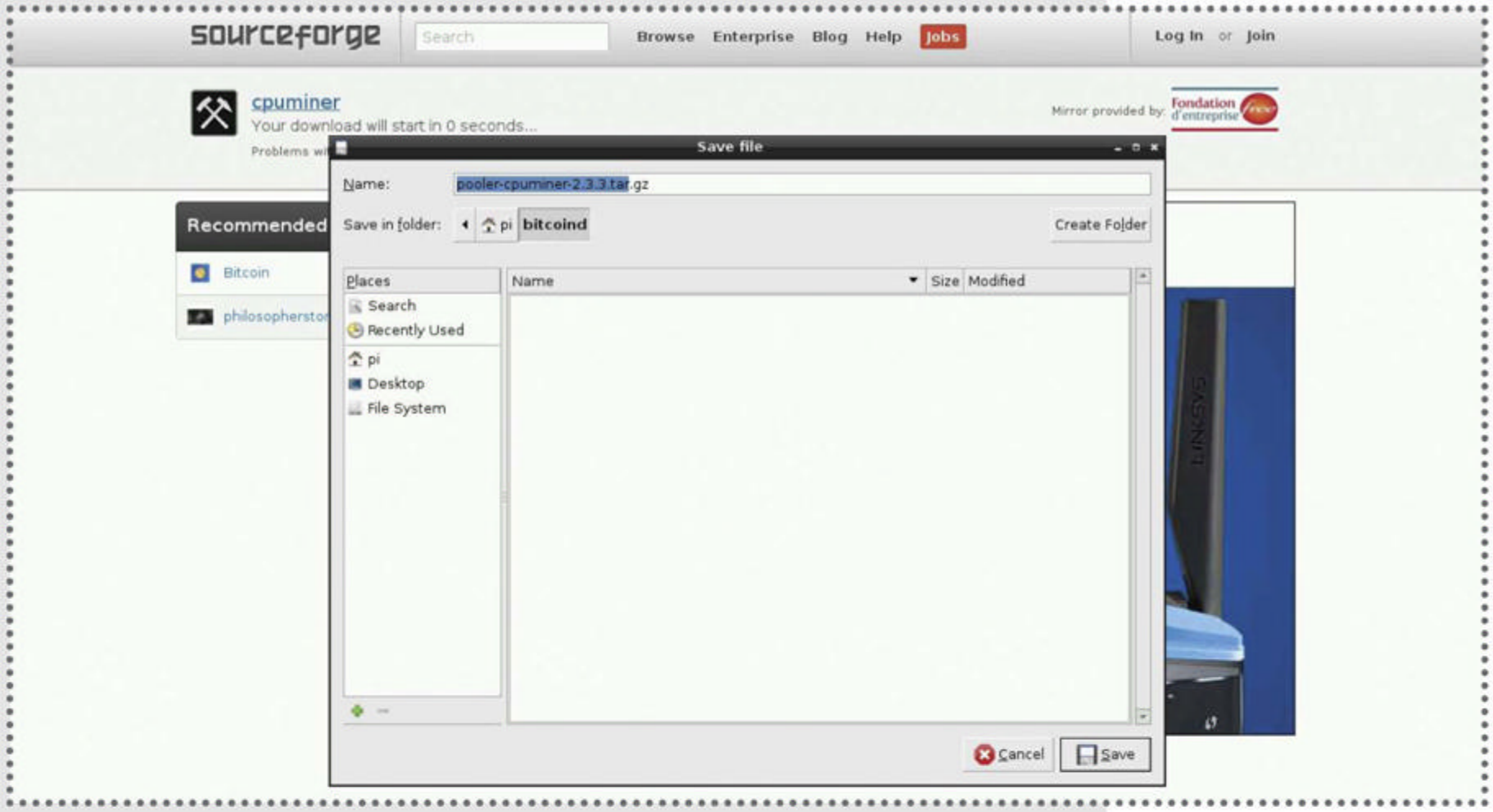
In order to make the building we'll be doing soon a little bit easier, it's best to create a directory for our files now. Still in the terminal, use:

```
$ mkdir bitcoind
```

After you've done that, move to the new directory with

```
$ cd bitcoind
```

"You're increasing the security of the Bitcoin system by logging transactions properly"



Above Cpuminer can also be used to mine Litecoins, the second biggest cryptocurrency after the Bitcoin

03 Get cpuminer

Head to **http://sourceforge.net/projects/cpuminer** for cpuminer and download the latest version of the source from there to this new directory. It will be the latest version with no OS attached to the title. At the time of writing this is pooler-cpuminer-2.3.3.tar.gz.

04 Extract the files

If you're in the desktop you can easily extract the files from the file manager. Otherwise go back to the terminal you've been working in and unpack it with:

```
$ tar -zxvf cpuminer-1.0.2.tar.gz
```

Follow up by moving into the new cpuminer folder with **cd**.

05 Build cpuminer

To build cpuminer we'll be using two very standard compiling commands: **./configure** and **make**.

Both may take a while to complete, with the **make** command likely to take the longest time. If you've followed the preceding steps properly, compiling should not be an issue for you.

06 Join a pool

Mining solo is generally considered a futile effort if you want to actually make Bitcoin out of the process. Joining a pool is the best way to do this and be involved with the community at the same time. Sign up to get a username, password and URL.

07 Start mining

With your worker details secure, go back to your Raspberry Pi and **cd** to the cpuminer folder again. To begin with cpu mining, enter the following command:

```
$ ./minerd --url [pool address] --userpass
[username]:[password]
```

Cpuminer will then show your hashrate as it goes, which will probably be a touch slow at this stage.

Left 'Slush's pool' was the world's first Bitcoin mining pool and has mined close to 1 million Bitcoins since 2010.

"Mining solo is generally considered a futile effort if you want to actually make Bitcoin out of the process. Joining a pool is the best way"

08 Increase mining speed

If you want to try and get a few more hashes per second from your Pi, you can always overclock it. In the terminal, open raspi-config and find the overclock option. You can increase it to whatever speed you wish but be aware that overclocking may seriously reduce your Pi's lifespan.

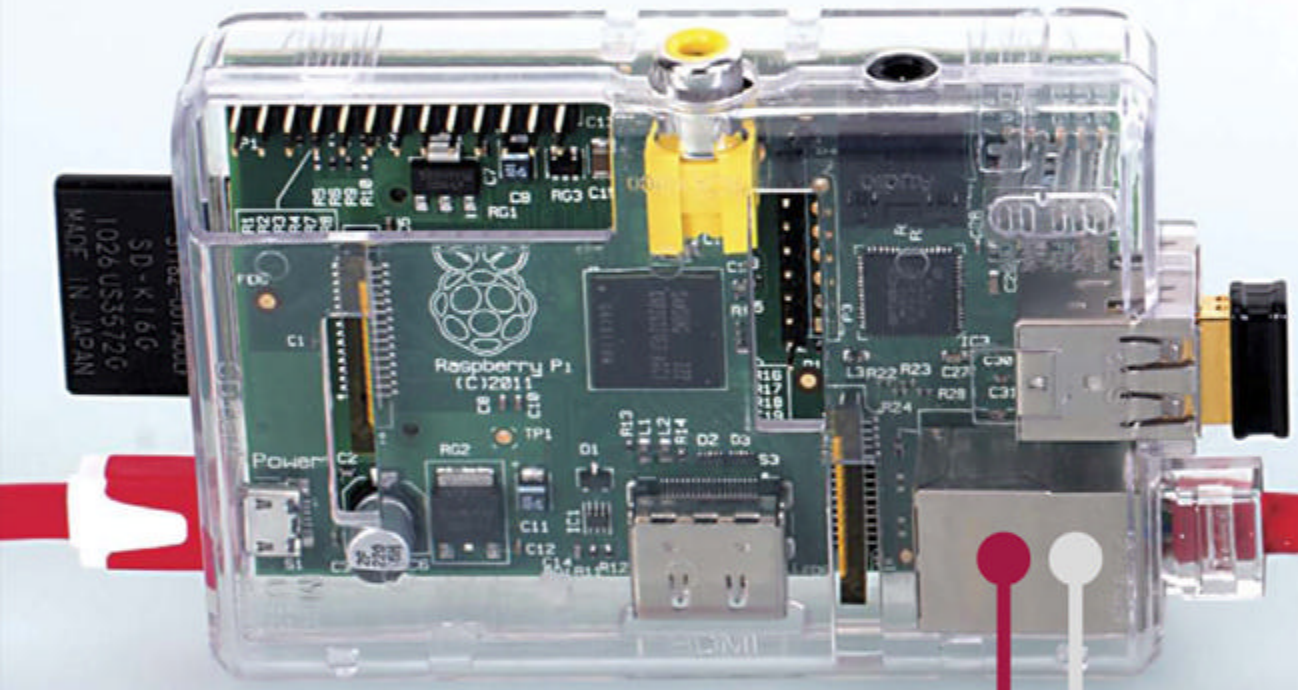
09 Future prospects

You're unlikely to make much out of a Raspberry Pi this way; if you want to make a serious amount of Bitcoins you'll need to increase your hashrate. You can expand your operation by attaching USB Bitcoin miners to the Raspberry Pi and powering it up exponentially.

Below Some of the newer USB miners are incredibly fast, but you'll still have to offset the cost of electricity consumption

Volunteer CPU power with BOINC Pi

Lend your Raspberry Pi's processing power to a good cause using BOINC, helping to cure disease along the way



"Protein folding for medical sciences, crunching Large Hadron Collider data and the search for extraterrestrial intelligence"

Some people manage to use their Raspberry Pis all the time. Whether they're powering a robot, an automated house, a media centre or even a normal desktop PC, there are some excellent ways to keep your Pi occupied. There are always a few neglected Pis though. If you don't fancy starting your own Bitcoin farm with our other tutorial this issue, you can always donate the processing power to a more worthy cause using BOINC, a computing volunteering service. These causes include protein folding for medical sciences, crunching Large Hadron Collider data and the search for extraterrestrial intelligence – all of which can benefit from the Raspberry Pi.

THE PROJECT ESSENTIALS

Latest Raspbian image
raspberrypi.org/downloads

BOINC
boinc.berkeley.edu/download all.php

A BOINC project
boinc.berkeley.edu/projects.php

01 Configure your Pi

Before we install BOINC, we need to configure the way Raspbian and your Raspberry Pi works. Open the terminal and use **sudo raspi-config** to enter the configuration options, then go to Advanced Options and find the Memory Split option. Change the number to 16 and hit OK followed by Finish to reboot.

> "Enter the configuration options, then go to Advanced Options and find the Memory Split option. Change the number to 16"

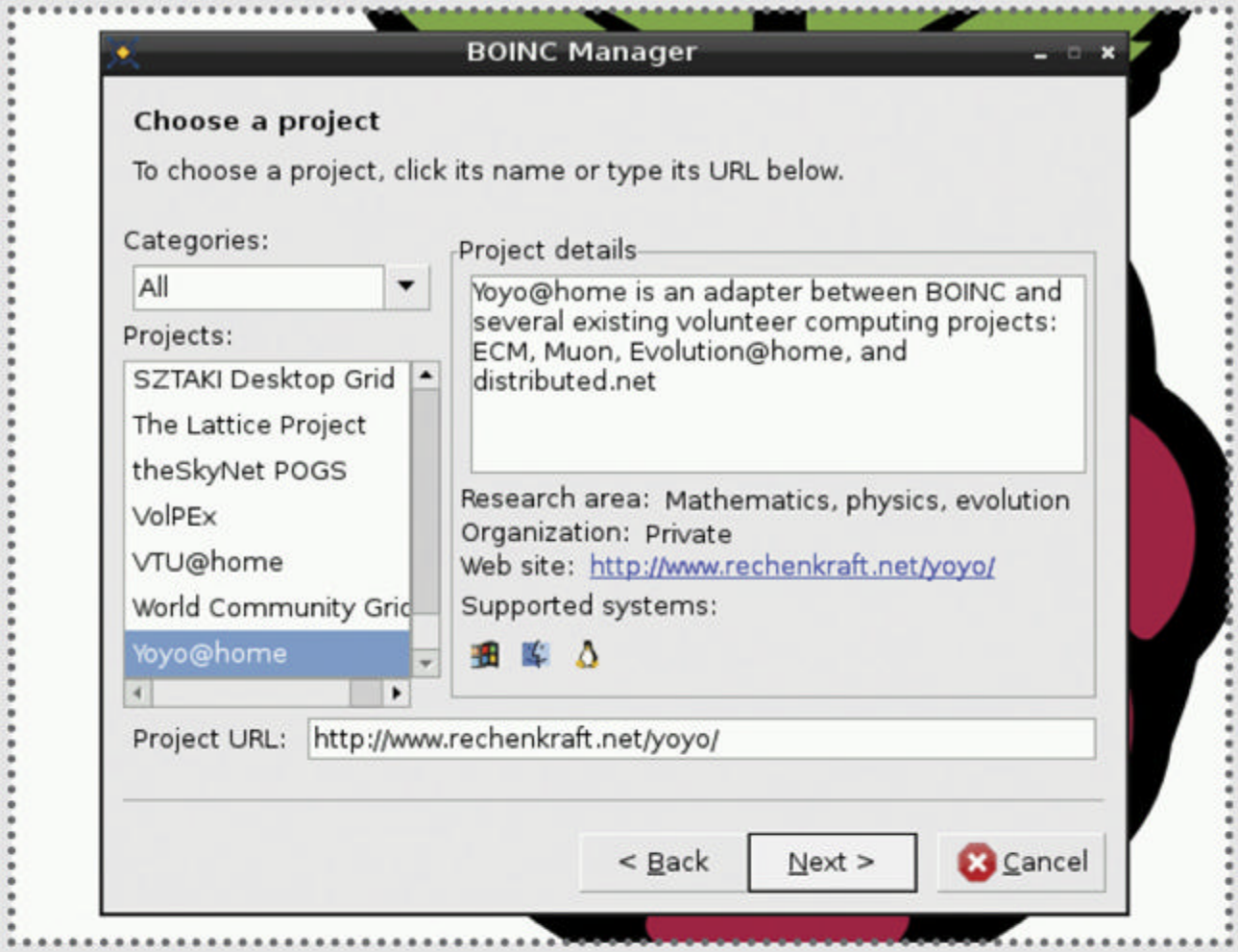
02 Install BOINC software

We can now install BOINC from Raspbian's repositories along with some other necessary packages to run it. Once you're back into Raspbian, open up the terminal again to install them all with:

```
$ sudo apt-get install boinc-manager boinc-client
```

03 First launch

If Raspbian is not already on the desktop environment, use **startx** to launch LXDE. Go to the Programs menu



Left All the projects you can donate computing power to are explained inside BOINC Manager

then System Tools to find BOINC Manager. Click on that to open it and begin the process of creating a new account and adding projects.

04 Add a project

Here we can add a new project to BOINC. If you haven't made a decision on what project to use yet, you will be able to look over all of the possibilities now. Make sure Add Project is selected and click Next. There are many categories you can choose from to narrow down your search; choose a project and click Next again.

05 Identify yourself

After clicking Next you'll need to create a BOINC account. You need a new account for each project; this allows you to have multiple systems donating to the same projects under your name. Create the

"You need a new account for each project; this allows you to have multiple systems donating to the same projects under your name"

account or login and click Finish once done – some projects will then ask you to name a machine on a separate webpage.

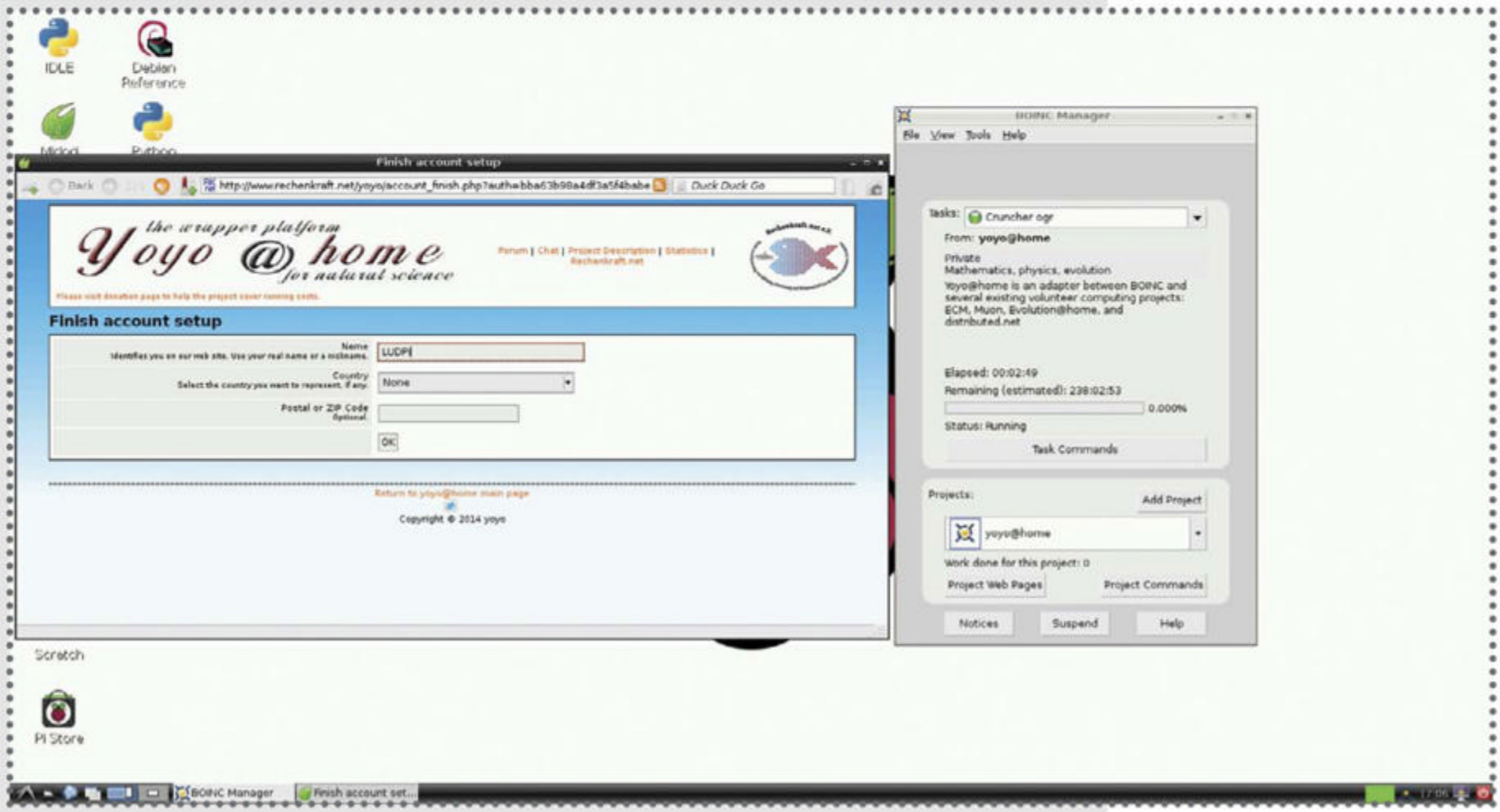
06 Configure BOINC

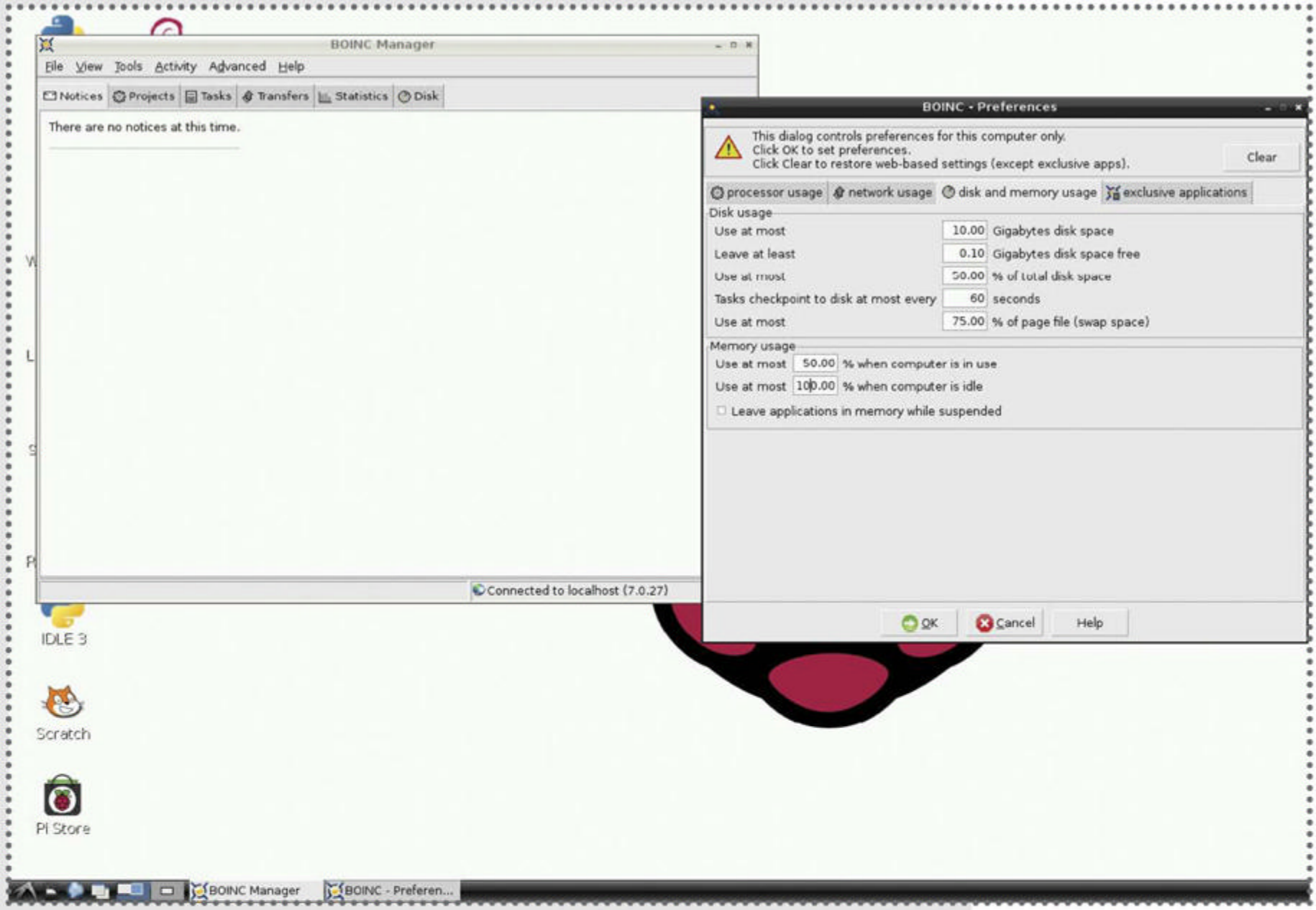
Click View on the top bar and select Advanced View to access more settings for BOINC. Go to Tools then Computing Preferences and select the 'Disk and memory usage' tab; from here you want to change Memory Usage to '100% while idle' to make the most of the Raspberry Pi.

07 Add more projects

Click View and Simple View to return to the original window. In the bottom section there's an Add Project button that allows you to not only add BOINC-approved projects, but also other projects with a BOINC link that are not necessarily included in the directory.

Below If you're asked to name your device, go for something clear in case you decide to donate with more computers later





Above There are plenty of useful settings to explore in the Preferences menu

08 Launch from command line

If you plan to keep your Raspberry Pi crunching data out of the way of your normal system, you might prefer to SSH in to the Pi. BOINC does have its own command line tool that can be accessed via:

```
$ boinccmd
```

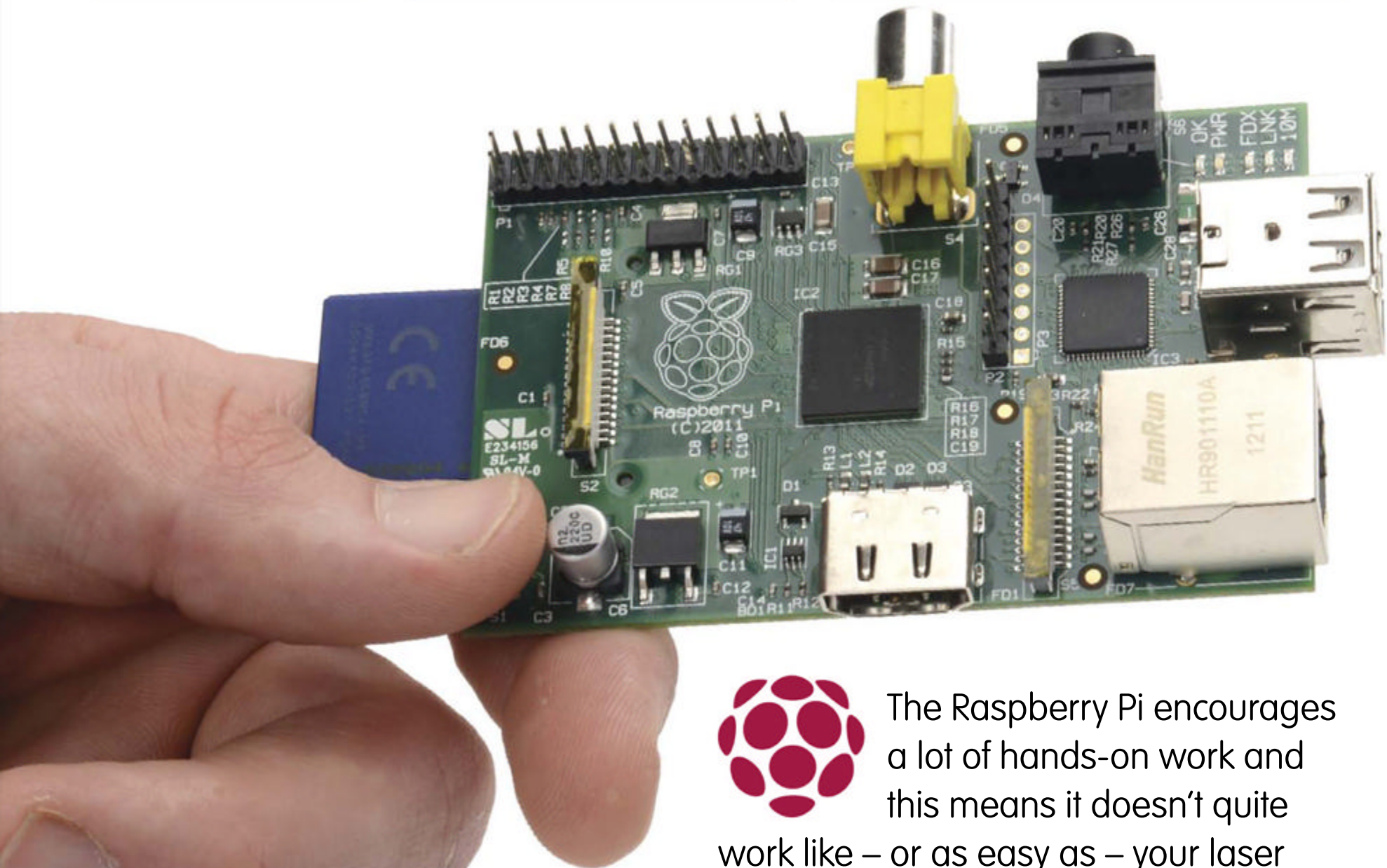
09 Crunch the numbers

BOINC will launch automatically when the Raspberry Pi is turned on, so find it a little space to stay appropriately powered and cool and let it do its work. You can add more projects in the future with the command line or by plugging it back into a monitor.

Talking Pi

Join the conversation at...

@linuxusermag | Linux User & Developer | @ RasPi@imagine-publishing.co.uk



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easy as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

Should I get a Raspberry Pi 2 or a B+?
Jake via Twitter

Good question! It comes down to how powerful you need your Pi to be. The new 2B has a faster processer (quad-core 900MHz versus the B+'s single-core 700MHz) and it also has twice as much RAM (1GB compared to 512MB). If you're going to be doing some proper computational work on your Pi then you'll definitely want the 2B, and it still costs just $35. The B+ has actually dropped in price to just $25, so if you're looking to have some fun with weekend projects – or even use several Pis as digital photo frames, for example – then the B+ is still a worthwhile buy.

I want to make an emulation station for my old games. Can you recommend any software?
Kirsten via Email

Well the RetroPie project is usually the first port of call for anyone looking to emulate their retro games library once the consoles themselves have worn out and fallen to bits – check out **http://bit.ly/1eio2BZ** to learn more about the setup. A simpler solution might be to try out the Lakka distro, which is incredibly user friendly – all you need to do is drop ROMs onto your SD card and plug in a controller. Head to **www.lakka.tv** to download the image.

Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

JUST A SCORE
WHAT'S YOUR JUST A SCORE?

Have you heard of Just A Score? It's a new, completely free app that gives you all the latest review scores. You can score anything in the world, like and share scores, follow scorers for your favourite topics and much more. And it's really good fun! >>

Can you recommend a good source of teaching material for the Raspberry Pi?
David via Email

The best place for educational resources is the official Raspberry Pi website – go to **www.raspberrypi.org/resources** and then click on the Teach link. There's a handful of work schemes available to download, including lesson plans and student worksheets, and there's also plenty more inspiration over in the Learn and Make sections of the site as well.

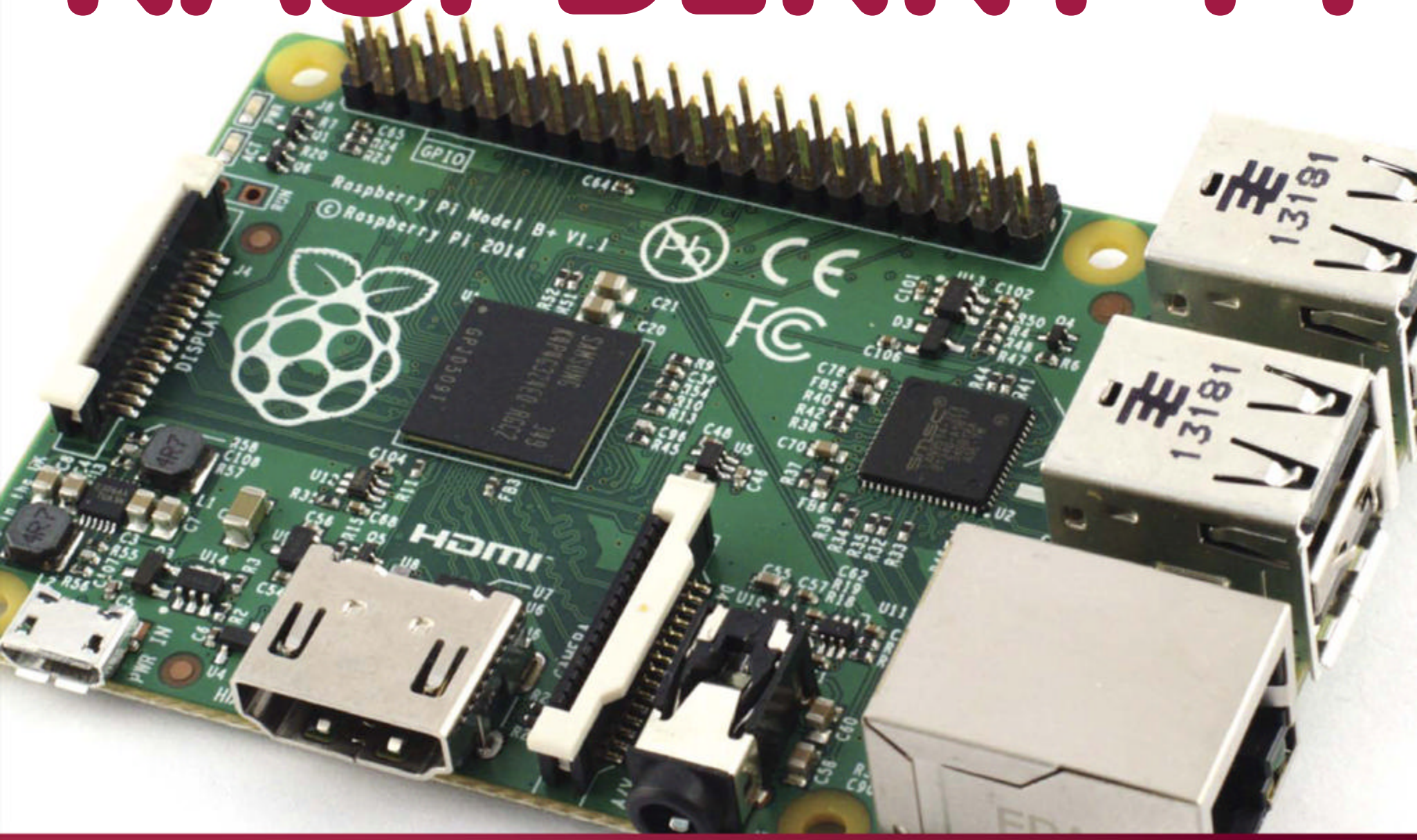
Can I still enter the Astro Pi competition?
Alice via Facebook

Yep – the primary school competition has closed but there's still time to enter the secondary school competition, and get your code running on a Pi aboard the ISS if you win. You will need an Astro Pi board to enter, and these were sent out to the early entrants during the first phase of the competition – however, at the time of writing, the official website says that more Astro Pi boards will be made available to schools and education groups and that details will be posted soon. So check out **http://bit.ly/1HwUMDY** – by now, you should be able to find out where to get hold of one. You then have until 29 June to submit your code and the winner will be announced in the Autumn. Good luck!

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

BUILD A SUPER RASPBERRY PI



Get this issue's source code at:

www.linuxuser.co.uk/raspicode